

EXHIBIT 1

Brandon H. Brown (SBN 266347)
Kyle Calhoun (SBN 311181)
KIRKLAND & ELLIS LLP
555 California Street
San Francisco, CA 94104
Telephone: (415) 439-1400
Facsimile: (415) 439-1500
Email: brandon.brown@kirkland.com
Email: kyle.calhoun@kirkland.com

Todd M. Friedman (admitted *pro hac vice*)
KIRKLAND & ELLIS LLP
601 Lexington Avenue
New York, NY 10022
Telephone: (212) 446-4800
Facsimile: (212) 446-4900
Email: todd.friedman@kirkland.com

Attorneys for Plaintiff RED HAT, INC.

UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN JOSE DIVISION

RED HAT, INC.,

Plaintiff,

v.

VIRTAMOVE, CORP.,

Defendant.

CASE NO. 5:24-CV-04740-PCP

SUPPLEMENTAL COMPLAINT

Jury Trial Demanded

Judge: Hon. P. Casey Pitts
Courtroom: 8, 4th Floor

NATURE OF THE ACTION

1. This is an action for declaratory relief arising under the patent laws of the United States.

2. Red Hat, Inc. (“Red Hat”) brings the instant action because Defendant VirtaMove, Corp. (“VirtaMove”) has commenced an aggressive litigation campaign against products relating to open source containerization and container orchestration technology. More specifically, VirtaMove has filed lawsuits against International Business Machines Corp. (“IBM”), Hewlett Packard Enterprise (“HPE”), Google, and Amazon.com Services LLC (“Amazon”), alleging that they infringe United States Patent Nos. 7,519,814 (the “’814 patent”) and 7,784,058 (the “’058 patent”) (collectively, the “Asserted Patents”). A true and correct copy of the operative complaints against IBM, HPE, Google, and Amazon are attached as Exs. A-D.

3. In these lawsuits, VirtaMove accused IBM, HPE, Amazon, and Google for use of their cloud and server computing products that offer containerization and container orchestration technology—including IBM’s Cloud Kubernetes Service, HPE Ezmeral Runtime Enterprise, Google’s Migration to Container, and Amazon Elastic Container Service (“ECS”) (collectively, the “Texas Cases Accused Products”). In its infringement contentions against IBM, HPE, and Google, VirtaMove identified entirely third-party, open source software—namely, Docker containers and the Kubernetes platform—as its basis for concluding that the Texas Cases’ Accused Products infringe. VirtaMove did not accuse or identify OpenShift, Red Hat’s containerization and container orchestration product, of infringing.

4. However, Red Hat’s OpenShift uses the same third-party Docker container and Kubernetes technology in its containerization and container orchestration operation. Because VirtaMove effectively concedes that mere use of Docker container and Kubernetes technology to implement containerization and container orchestration is sufficient to infringe its ’814 and ’058 patents, there is a cloud over Red Hat’s products; a threat to Red Hat’s business, its relationships with its customers and partners, and its sales of OpenShift; and a justiciable controversy between Red Hat and VirtaMove.

5. Therefore, Red Hat requests relief as follows: a declaratory judgment that Red Hat’s products do not infringe the Asserted Patents because they do not meet each and every limitation of any asserted claim.

THE PARTIES

6. Plaintiff Red Hat is a corporation organized and existing under the laws of the state of Delaware, with its principal place of business at 100 East Davie Street, Raleigh, NC 27601. Red Hat is a well-known software company that provides open source software products, including its OpenShift product, to enterprise customers. Red Hat is a subsidiary company of IBM after being acquired by IBM in 2019 but continues to operate as a separate entity.

7. Defendant VirtaMove is a corporation organized and existing under the laws of Canada, with its principal place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. *See* Ex. E (VirtaMove Corporate Profile). VirtaMove is formerly known as Appzero Software Corp. (“Appzero”), which was established in 2010. *Id.* VirtaMove also maintains an office location at 300 Brickstone Square, Suite 201, Andover, MA 01810.

JURISDICTIONAL STATEMENT

8. This action arises under the Declaratory Judgment Act, 28 U.S.C. § 2201, under the patent laws of the United States, 35 U.S.C. §§ 1–390.

9. This Court has subject matter jurisdiction over this action under 28 U.S.C. §§ 1331, 1338(a) and 2201(a).

10. This Court has personal jurisdiction over VirtaMove. Among other things, VirtaMove has continuous and systematic business contacts with Northern California.

11. For example, on information and belief, on November 13, 2013, AppZero, the predecessor to VirtaMove, partnered with HP Cloud Services for its products as part of its “Premier Partner Program.”

Image is 100s of times faster to move and is available for Windows.

VirtaMove, Corp. v. Hewlett Packard Enterprise Co., No. 2:24-cv-00093-JRG, Dkt. 40, ¶¶ 10–11 (E.D. Tex. Feb. 9, 2024); *see also* <https://icloud.pe/blog/appzero-adds-three-premier-partners-and-sees-significant-growth/>;

https://www.prweb.com/releases/appzero_adds_three_premier_partners_and_sees_significant_growth/prweb11310700.htm.

1 12. This partnership took place in 2013, prior to the split of Hewlett-Packard into two separate
2 entities, HP Inc. and Hewlett Packard Enterprise (“HPE”). Thus, at the time, this partnership was with
3 HP Inc.

4 13. HP is, and was in 2013, a California company with its headquarters in Palo Alto, CA, in the
5 Northern District of California. See [https://www.hp.com/us-en/hp-information/cwc/ww-briefing-](https://www.hp.com/us-en/hp-information/cwc/ww-briefing-center.html)
6 [center.html](https://www.hp.com/us-en/hp-information/cwc/ww-briefing-center.html). Moreover, based on information and belief, VirtaMove representatives met with HP
7 representatives in California to discuss and demonstrate the AppZero technology that VirtaMove claims
8 is covered by the ’814 and ’058 patents. *VirtaMove, Corp. v. Hewlett Packard Enterprise Co.*, No. 2:24-
9 cv-00093-JRG, Dkt. 40, ¶¶ 10–11 (E.D. Tex. May 13, 2024).

10 14. As another example, in 2015, 2020, and 2021, AppZero met with representatives of Google
11 “for the purposes of partnering with VirtaMove, demonstrating AppZero, training Google on how to use
12 AppZero, allowing Google to run and evaluate AppZero, discussing integration of AppZero into Google
13 cloud, and sharing materials about how AppZero works,” and that Google allegedly “would have learned
14 that AppZero was patented” at that time. See *VirtaMove, Corp. v. Google LLC*, No. 7:24-cv-00033-DC-
15 DTG, Dkt. 27, ¶ 10 (W.D. Tex. May 21, 2024). Google is a California company with its headquarters in
16 Mountain View, CA, in the Northern District of California. *Id.*, ¶ 6. Based on information and belief, this
17 demonstration of the VirtaMove technology purportedly underlying the Asserted Patents and discussion
18 meetings between AppZero and Google took place in the Northern District of California.

19 15. Likewise, VirtaMove has established and maintains strategic partnerships and relationships
20 with companies in Northern California. For example, on April 15, 2020, VirtaMove announced a
21 “Strategic Partnership” with the company CloudPhysics “to Accelerate Application Modernization.” See
22 [https://virtamove.com/blog/virtamove-and-cloudphysics-co-announce-strategic-partnership-to-](https://virtamove.com/blog/virtamove-and-cloudphysics-co-announce-strategic-partnership-to-accelerate-application-modernization/)
23 [accelerate-application-modernization/](https://virtamove.com/blog/virtamove-and-cloudphysics-co-announce-strategic-partnership-to-accelerate-application-modernization/). That same announcement explains that CloudPhysics is
24 “[h]eadquartered in Santa Clara, CA,” in the Northern District of California. On information and belief,
25 through CloudPhysics’ use of VirtaMove’s application virtualization products as part of the strategic
26 partnership and offering to joint customers, CloudPhysics was granted an express or implied license to
27 use VirtaMove’s application virtualization patents, including the ’814 and ’058 patents.

1 16. VirtaMove also purposefully utilizes California servers for VirtaMove’s product V-Migrate
2 to use, sell, offer for sale, license, and/or distribute V-Migrate for and to California customers. For
3 example, VirtaMove markets V-Migrate to customers as a solution intended “to move legacy workloads
4 to the Cloud,” and specifically the AWS Cloud, provided by Amazon. *See*
5 <https://virtamove.com/blog/virtamove-partners-with-aws-launches-saas-in-aws-marketplace/>. The AWS
6 US West (Northern California) Region server location is in the Northern District of California. *See*
7 https://aws.amazon.com/about-aws/global-infrastructure/regions_az/. On information and belief,
8 VirtaMove knows that these servers are located in the Northern District of California and purposefully
9 makes use of these server locations to use, sell, offer for sale, license, and/or distribute V-Migrate for and
10 to customers in California.

11 17. On information and belief, AppZero also partnered with the company OpSource, Inc. to host
12 VirtaMove’s application virtualization products. *See* [https://www.varinsights.com/doc/opsources-](https://www.varinsights.com/doc/opsources-introduces-opsources-partner-0001)
13 [introduces-opsources-partner-0001](https://www.varinsights.com/doc/opsources-introduces-opsources-partner-0001). OpSource maintains server locations in Santa Clara, CA, in the
14 Northern District of California, used by VirtaMove as part of this partnership. *See*
15 <https://www.datacenterdynamics.com/en/news/opsources-launches-santa-clara-data-center/>.

16 18. VirtaMove has additional contacts with the forum, including contacts relating to its efforts to
17 license, enforce, and sell products allegedly embodying its patents. For example, members of
18 VirtaMove’s Board are based in Northern California, including at least Scott Munro, who is also a director
19 of VirtaMove with fiduciary duties and obligations. *See* Ex. E (VirtaMove Corporate Profile). The
20 VirtaMove representative who purportedly met with the defendants to “discuss and demo AppZero and
21 its technology for either partnership use, distribution, and/or investment” is, on information and belief,
22 Greg O’Connor, who at the time was the CEO of AppZero. Mr. O’Connor resides in the Northern District
23 of California. And the attorneys and law firm VirtaMove hired to enforce its patents are located in
24 California.

25 19. Red Hat OpenShift, the product imminently threatened with patent infringement litigation,
26 was developed, in part, in the Northern District of California, and many of Red Hat’s OpenShift customers
27 are also based in the Northern District of California. And as explained above, VirtaMove’s infringement
28 allegations are directed primarily to the use of technologies, such as Kubernetes and Docker, that were

likewise developed and that are maintained in the Northern District of California. In particular, California-based Google launched Kubernetes as an open source project based on Google's work on Borg, which was likewise developed at Google in California. Kubernetes was placed under the governance of the Cloud Native Computing Foundation ("CNCF"), which is a part of the Linux Foundation, both located in San Francisco, California. Docker containers were developed in California by Docker, Inc., which is located in San Francisco and Palo Alto, California. In addition, Red Hat implements its ContainerD and CRI-O container engines in its current version of OpenShift. CRI-O is an open source container solution specifically designed to work with Kubernetes and is maintained by the CNCF, just as Kubernetes is, located in San Francisco, CA. ContainerD is an open source container runtime originally developed by Docker but today is also maintained by the CNCF. On information and belief, VirtaMove knows that these companies and technologies are based in California.

20. VirtaMove is also registered to do business in California and has a registered corporate agent in California. *See* Ex. F (CA Sec. of State).

21. Venue is proper in this District under 28 U.S.C. §§ 1391(b), (c), because a substantial part of the events giving rise to VirtaMove's claims occurred in this District, and because VirtaMove is subject to personal jurisdiction here.

FACTUAL BACKGROUND

RED HAT AND ITS OPEN SOURCE SOFTWARE

22. Red Hat is a leading contributor to the open source software community. Although Red Hat makes software available under open source licenses, Red Hat derives revenue from aggregating, certifying, testing, enhancing, packaging, maintaining, supporting, and influencing the future direction of the software, among other value-added offerings. Red Hat remains committed to the open source development model, and many thousands of Red Hat's employees have contributed and continue to contribute to the open source software ecosystem, including by developing and releasing code under open source licenses. Red Hat believes that the open source development and licensing model offers important advantages for its customers over proprietary software development and licensing models.

23. Open source software is software in which the source code is made available to users for inspection, modification, and distribution. Generally, when a computer program is authored, the

1 programmer writes code in a human-readable programming language called “source code.” The source
2 code can be compiled into another form, called “object code,” that is executable by a computer
3 microprocessor. With open source software, the source code itself is made available to the recipient under
4 conditions set forth in an accompanying license, which grants relatively broad rights to recipients to use,
5 copy, modify, and distribute the software, but may also limit the ways in which the code or derivative
6 works of the code can be distributed so as to benefit the broader developer community.

7 24. The benefits of the open source model are widely recognized. For instance, in 2008, the
8 Federal Circuit commented “[o]pen source licensing has become a widely used method of creative
9 collaboration that serves to advance the arts and sciences in a manner and at a pace that few could have
10 imagined just a few decades ago.” *Jacobsen v. Katzer*, 535 F.3d 1373, 1379 (Fed. Cir. 2008).

11 25. One common open source software license is the GNU General Public License (“GPL”). The
12 GPL permits access to human-readable software source code and provides relatively broad rights for
13 licensees to use, copy, modify, and distribute open source software.

14 26. Red Hat is also one of the largest corporate contributors to the Linux kernel, which is a
15 collection of programs at the heart of the Linux operating system. Through this work, Red Hat developed
16 Red Hat Enterprise Linux (“RHEL”), which is a distribution of the Linux operating system designed for
17 enterprise environments. One of the key features of RHEL is its subscription-based model, which provides
18 customers with access to software updates, security patches, and technical support from Red Hat. In
19 November 2019, Red Hat released version 8.1 of RHEL. RHEL contains numerous components which
20 are licensed under the GPL, and other open source licenses.

21 27. For instance, Red Hat OpenShift is a family of containerization software products. One such
22 product is the OpenShift Container Platform, which is a hybrid cloud platform built around Linux
23 containers orchestrated and managed by Kubernetes on a foundation of Red Hat Enterprise Linux.
24 OpenShift is an open source software product made available to third-parties through various open source
25 licenses.

26 28. Because Red Hat provides its products as open source software programs, Red Hat maintains
27 an open source assurance program. When a client obtains certain Red Hat open source software, they
28 enter into the Red Hat Open Source Assurance Agreement. This agreement is designed to protect

1 customers developing and deploying Red Hat solutions. A true and correct copy of this agreement is
2 attached hereto as Exhibit G.

3 *CONTAINERIZATION TECHNOLOGY*

4 29. Containerization software is a technology that allows a user to package and run an application
5 in a portable environment called a container. The container runs as an isolated process on the host
6 operating system, which ensures that the application runs consistently regardless of the environment in
7 which it is deployed.

8 30. The idea for dividing and separating systems and applications within a computing
9 environment is not new. Back in the 1970s, LPARs (Logical Partitions) were developed at IBM as a
10 virtualization technology for mainframe computers, allowing a single physical system to be divided into
11 multiple logical servers. Each of the resulting divided systems ran its own operating system and
12 applications. Containers, similarly, provide a means of virtualization that package cloud-native
13 applications across different computing environments. LPARs were, and are, the foundational technology
14 for providing isolation between applications running on the same physical or virtual system, providing the
15 user efficiency and flexibility.

16 31. Early iterations of containerization software products include Thinstall (now known as
17 ThinApp), FreeBSD, Virtuozzo, and Solaris, all developed in NDCA, and Zap, developed at Columbia
18 University.

19 32. In the early 2000's, Google engineers in Mountain View, CA, developed the project Borg,
20 which outlined the principles and design of a container orchestration system. In 2014, Google announced
21 the launch of Kubernetes as an open source project based on Google's work on Borg. Shortly thereafter,
22 Kubernetes was placed under the governance of the Cloud Native Computing Foundation ("CNCF"),
23 which is a part of the Linux Foundation, both located in San Francisco, CA. Like Red Hat OpenShift,
24 Kubernetes is an open source software program available under various open source licenses and as such,
25 is the result of collaborative efforts of the open source community.

26 33. Around this same time, researchers and engineers at the University of California at Berkeley
27 ("Cal") were developing another container orchestration system called Apache Mesos. Mesos also is now
28

1 an open source software program available under the Apache License 2.0, a widely-used open source
2 license, and as such, is the result of collaborative efforts of the open source community.

3 34. Around this time, Docker, Inc. developed a container orchestration platform called Swarm.
4 The Docker software was first publicly released in Santa Clara, CA in 2013. Docker, Inc. is located in
5 Palo Alto, CA.

6 35. To make use of a container orchestration platform like Kubernetes, Mesos, and Swarm, a
7 software for developing and running applications in containers is required. Popular software programs
8 include Docker, Garda, and Moby (all developed by Docker, Inc. in Palo Alto, California), ContainerD
9 (originally developed by Docker, Inc. but later placed under governance of the CNCF as an open source
10 project), and Red Hat OpenShift.

11 36. Since their inception, Red Hat engineers contributed and continue to contribute to the open
12 source code underlying containerization software products such as Kubernetes and Docker.

13 37. OpenShift was developed by Red Hat as the market began adopting containers and
14 Kubernetes. OpenShift uses Kubernetes as its underlying container orchestration system and Red Hat
15 Enterprise Linux as its operating system. Today, OpenShift is a leading enterprise Kubernetes platform,
16 providing organizations with a solution for building, deploying, and managing cloud-native applications
17 at scale across on-premises data centers, public clouds, and edge environments.

18 38. OpenShift version 1 was released in November 2010. Red Hat released OpenShift as an open
19 source project in May 2012.

20 39. Originally, OpenShift used a custom approach to implementing Linux containers, and did so
21 through version 2 of the product. After version 2, OpenShift shifted to use of Docker containers, along
22 with Kubernetes as its container orchestration system. In version 4 of the product (the current version),
23 Red Hat implemented its ContainerD, CRI-O, and Podman container engines in OpenShift.

24 40. CRI-O is an open source container solution specifically designed to work with Kubernetes. It
25 was originally developed by Red Hat but today is now maintained by the CNCF, just as Kubernetes is,
26 located in San Francisco, CA.

27 41. Podman is likewise an open source container management tool that was developed by Red
28 Hat.

42. ContainerD is an open source container runtime originally developed by Docker but today is now maintained by the CNCF. ContainerD is widely used for container orchestration platforms like Kubernetes.

43. IBM offers a service to third-party enterprises to manage containerization of their applications called IBM Cloud Kubernetes Service (“IBM Kubernetes”). Initially, IBM partnered with Docker, Inc. to bring containerization technologies to enterprise clients and integrated Docker into its platform. After Docker moved to a more proprietary route, IBM began using Kubernetes and other container platforms such as through the Open-Container Initiative (“OCI”).

44. IBM Kubernetes is a service for businesses to manage their containerization, utilizing software developed by third parties such as Kubernetes (developed by CNCF), Docker (developed by Docker, Inc.), and other open source container projects.

RELEVANT RELATED LITIGATION

45. On January 31, 2024, VirtaMove sued IBM in the Eastern District of Texas alleging infringement of the Asserted Patents. See *VirtaMove, Corp. v. Int’l Bus. Machines Corp.*, No. 2:24-cv-00064-JRG (W.D. Tex. Jan. 31, 2024); Ex. I.

46. The ’814 patent is titled “System for Containerization of Application Sets.” The ’814 patent is based on application No. 10/939,903 filed on September 13, 2004 and supposedly claims priority to provisional application No. 60/502,619, filed on September 15, 2003. A true and correct copy of the ’814 patent is attached hereto as Exhibit J.

47. The ’058 patent is titled “Computing System Having User Mode Critical System Elements as Shared Libraries.” The ’058 patent is based on application No. 10/946,536 filed September 21, 2004 and supposedly claims priority to provisional application No. 60/504,213, filed on September 22, 2003. A true and correct copy of the ’058 patent is attached hereto as Exhibit K.

48. The named inventors for both Asserted Patents are Donn Rochette, Paul O’Leary, and Dean Huffman. On information and belief, Mr. Rochette currently resides in Ohio and Mr. O’Leary and Mr. Huffman in Ottawa, Canada. The Asserted Patents are assigned to Trigence Corp., an Ottawa, Canada corporation. On information and belief, Trigence was later acquired by or merged with Appzero, which later became VirtaMove.

1 49. In its May 29, 2024 Second Amended Complaint against IBM, VirtaMove alleged IBM's
2 Kubernetes infringed the Asserted Patents, focusing exclusively on functionality provided by third-party
3 software Kubernetes and Docker. A true and correct copy of those infringement contentions is attached
4 hereto as Ex. A at Exhibits 2 and 4.

5 50. Around the same time VirtaMove sued IBM, VirtaMove also sued Amazon, Google, and HPE
6 alleging infringement of the same Asserted Patents. On January 26, 2024, VirtaMove sued Amazon in
7 WDTX alleging infringement by Amazon's AWS End-of-Support Migration Program ("EMP") and
8 Elastic Container Service ("ECS"). *VirtaMove, Corp. v. Amazon.com, Inc.*, No. 7:24-cv-00030, Dkt. 1
9 (W.D. Tex. Jan. 26, 2024) ("*Amazon Case*"). On January 31, 2024, VirtaMove sued Google in WDTX
10 alleging infringement by Google's Migrate to Containers service. *VirtaMove, Corp. v. Google LLC*, No.
11 7:24-cv-00033, Dkt. 1 (W.D. Tex. Jan. 31, 2024) ("*Google Case*"). And on February 9, 2024, VirtaMove
12 sued HPE in EDTX alleging infringement by HPE's Ezmeral Runtime Enterprise. *VirtaMove, Corp. v.*
13 *Hewlett Packard Enterprise Co.*, No. 2:24-cv-00093-JRG, Dkt. 1 (E.D. Tex. Feb. 9, 2024) ("*HPE Case*").
14 All of VirtaMove's infringement allegations in these matters are based on the use of Kubernetes and
15 Docker containers.

16 51. At a high-level, and based on Red Hat's understanding of VirtaMove's contentions without
17 conceding the scope or functionality of the Asserted Patents, VirtaMove's infringement allegations are
18 directed at two features: (1) Containerizing applications and their dependencies; and (2) Deploying the
19 resulting containers across a network infrastructure. Red Hat OpenShift uses the same open source
20 components accused of infringement in IBM, HPE, Amazon, and Google's products—Docker containers
21 (or containers generally) and Kubernetes. VirtaMove is aware of this. Indeed, in a June 8, 2020 blog post
22 on VirtaMove's own website, VirtaMove explains that OpenShift is "an evolution of the Kubernetes
23 Enterprise 1.17 platform. OpenShift Container Platform is a cloud development Platform as a Service
24 (PaaS) built around Docker containers. The platform is orchestrated and managed by Kubernetes on a
25 foundation of Red Hat Enterprise Linux. It offers a cloud foundation for building, deploying, and scaling
26 new containerized applications."

Red Hat OpenShift 4.4 Opens a World of Possibility

In Spring 2020, Red Hat released OpenShift 4.4., an evolution of the Kubernetes Enterprise 1.17 platform. OpenShift Container Platform is a cloud development Platform as a Service (PaaS) built around Docker containers. The platform is orchestrated and managed by Kubernetes on a foundation of Red Hat Enterprise Linux. It offers a cloud foundation for building, deploying, and scaling new containerized applications. It's particularly attractive for IT shops interested in developing cloud-enabled micro services.

Red Hat believes that companies need faster and more widespread access to essential apps and services. IT departments are pressured to become ever more agile to deliver innovative solutions. According to Red Hat, this landscape translates to "container and Kubernetes-powered open hybrid cloud". A preferred path in the enterprise cloud adoption journey is "application platforms supporting an architecture that gives developers a wide choice of components across hybrid cloud infrastructure."

At VirtaMove, we're also intrigued by new applications such as OpenShift, believing that we can use them to containerize legacy applications and retro fit them for load balancing and service architectures.

Containerization boasts many benefits, chiefly among these accelerating the development and deployment of apps on cloud infrastructure. Both Kubernetes and OpenShift promise easy container management.

See <https://virtamove.com/blog/shift-to-the-future-now/>.

52. Furthermore, VirtaMove has previously recognized that Red Hat OpenShift "is built around Docker containers" and "offers a cloud foundation for building, deploying, and scaling new containerized applications." <https://virtamove.com/blog/shift-to-the-future-now/>. Indeed, "building, deploying, and scaling new containerized applications" is not unique to Red Hat OpenShift, but rather a central feature in each of the products VirtaMove has accused of infringing.

53. VirtaMove's infringement contentions against IBM, HPE, Google, and Amazon could have just as easily been levied against Red Hat OpenShift without altering any of their substance.

54. For example, VirtaMove's infringement contentions against all four of the Defendants identify the use of Docker containers—specifically, the use of Dockerfiles and Docker images to create Docker containers—as infringing the Asserted Patents. Red Hat's OpenShift likewise offers Docker containers by building container images from Dockerfiles to create Docker images.

Building a simple container

You have an idea for an application and you want to containerize it.

First you require a tool for building a container, like buildah or docker, and a file that describes what goes in your container, which is typically a *Dockerfile*.

Next, you require a location to push the resulting container image so you can pull it to run anywhere you want it to run. This location is a container registry.

Some examples of each of these components are installed by default on most Linux operating systems, except for the *Dockerfile*, which you provide yourself.

The following diagram displays the process of building and pushing an image:

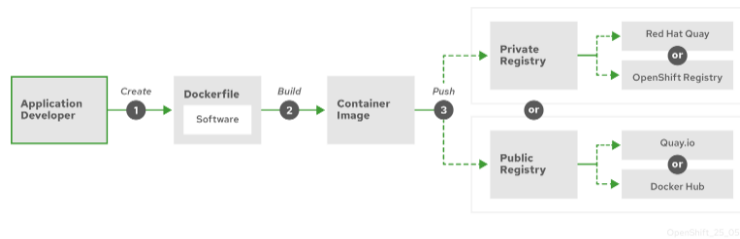


Figure 1. Create a simple containerized application and push it to a registry

<https://docs.openshift.com/container-platform/4.15/architecture/understanding-development.html>

55. As another example, for the limitation of the '814 patent requiring “storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one the servers,” based on Red Hat’s understanding of VirtaMove’s contentions, without conceding the scope or functionality of the Asserted Patents, VirtaMove relies on IBM’s use of application containers, which are based on Docker images created from Dockerfiles, in a Kubernetes environment. *See, e.g., Ex. L* (VirtaMove’s ’814 IBM Infringement Chart) at 9-22 (“For example, IBM Cloud Kubernetes stores **application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent storage available to each node running the application.** The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network. Each container includes the application software as well as a Linux user space required to execute the application, for example libc/glibc and other shared libraries, configuration files, etc. necessary for the application. For example, the container includes a base OS image, provided by IBM or by a third party, such as a CentOS, RHEL, or Ubuntu base image. The container is compatible with the host kernel, for example because the

1 container libraries are linked against the Linux kernel, and the supported host operating systems also use
 2 the Linux kernel, which has a stable binary interface.”) (emphasis added); Ex. C at Exhibit 2 (Google ’814
 3 Infringement Chart) at 5-7 (identifying the use of containers); Ex. B at Exhibit 2 (HPE ’814 Infringement
 4 Chart) at 4-8 (identifying the use of Docker containers in a Kubernetes environment). While Red Hat
 5 disputes that these features (and all of its features) infringe VirtaMove’s patents, all of the IBM features
 6 accused by VirtaMove are likewise offered in/supported by Red Hat OpenShift. With respect to Amazon,
 7 VirtaMove identifies “packaging of the application and its dependencies” and “migration [of that package]
 8 to [a new] environment.” *See* Ex. D at Exhibit 2 (Amazon ’814 Infringement Chart) at 4-6. Again, these
 9 identified Amazon features are also offered in/supported by Red Hat OpenShift. And just as that
 10 “package” is migrated to a new environment, containers in Red Hat OpenShift can be migrated to new
 11 environments. [https://docs.redhat.com/en/documentation/openshift_container_platform/4.15/html-](https://docs.redhat.com/en/documentation/openshift_container_platform/4.15/html-single/migration_toolkit_for_containers/index#about-mtc)
 12 [single/migration_toolkit_for_containers/index#about-mtc](https://docs.redhat.com/en/documentation/openshift_container_platform/4.15/html-single/migration_toolkit_for_containers/index#about-mtc).

13 56. As another example, for the limitation of the ’814 patent requiring “containers of application
 14 software excluding a kernel,” based on Red Hat’s understanding of VirtaMove’s contentions, without
 15 conceding the scope or functionality of the Asserted Patents, VirtaMove relies on the Texas Cases Accused
 16 Products’ use of containerization technology. *See, e.g.,* Ex. L (VirtaMove’s ’814 IBM Infringement
 17 Chart) at 22-24 (identifying the use of containers); Ex. C at Exhibit 2 (Google ’814 Infringement Chart)
 18 at 9-10 (identifying the use of containers in a Kubernetes environment); Ex. B at Exhibit 2 (HPE ’814
 19 Infringement Chart) at 9-10 (identifying the use of Docker containers); Ex. D at Exhibit 2 (Amazon ’814
 20 Infringement Chart) at 4-6 (identifying “packaging of the application and its dependencies” and
 21 “migration [of that package] to [a new] environment,” which describe the function of containers and
 22 Kubernetes).

23 <https://www.ibm.com/topics/containerization>

24 <https://ibm.github.io/kube101/>

25 Ex. L (VirtaMove’s ’814 IBM Infringement Chart) at 26-27 (identifying the use of containers). Red Hat
 26 OpenShift uses the same containerization technology that VirtaMove accuses of infringement. *See, e.g.,*
 27 https://docs.openshift.com/container-platform/4.15/openshift_images/index.html (“The basic units of
 28 OpenShift Container Platform applications are called containers.”).

57. As yet another example, in the limitation of the '814 patent requiring “wherein each of the containers has a root file system that is different from an operating system’s root file system[,]” based on Red Hat’s understanding of VirtaMove’s contentions, without conceding the scope or functionality of the Asserted Patents, VirtaMove relies on the Texas Cases Accused Products’ use of Dockerfiles to build Docker images comprising an application and its dependencies, which can then run in a Kubernetes environment as a container, as well as IBM’s use of Open Container Initiative (“OCI”) images. *See, e.g.*, Ex. L (VirtaMove’s ‘814 IBM Infringement Chart) at 36-46; Ex. C at Exhibit 2 (Google ‘814 Infringement Chart) at 14-15 (identifying the use of Docker containers and Kubernetes); Ex. B at Exhibit 2 (HPE ‘814 Infringement Chart) at 12 (identifying the use of Docker containers); Ex. D at Exhibit 2 (Amazon ‘814 Infringement Chart) at 4-6 (identifying “packaging of the application and its dependencies” and “migration [of that package] to [a new] environment,” which describe the function of containers and Kubernetes), 11 (“The package contains both the file and registry data of the packaged application, and the EMP binaries and configuration files that are required to deploy and run the packaged application.”). The IBM features accused by VirtaMove are features offered in and supported by Red Hat OpenShift. *See, e.g.*, Ex. H https://docs.openshift.com/container-platform/4.15/openshift_images/index.html (“Containers in OpenShift Container Platform are based on OCI- or Docker-formatted container images . . . You can use . . . [D]ocker CLI directly to build images.”); <https://docs.openshift.com/container-platform/4.15/architecture/understanding-development.html> (describing the process of building a container, which includes the steps of creating a Dockerfile and building a container image using Docker commands).

58. As another example, for the limitation of the '058 patent requiring “a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode[,]” based on Red Hat’s understanding of VirtaMove’s contentions, without conceding the scope or functionality of the Asserted Patents, VirtaMove relies on the Texas Cases Accused Products’ use of Dockerfiles to build Docker images comprising an application and its dependencies, which can later be deployed in a Kubernetes environment as containers, as well as IBM’s use of OCI images. *See, e.g.*, Ex. L (VirtaMove’s ‘058 IBM Infringement Chart) at 5-18; Ex. C at Exhibit 4 (Google ‘058 Infringement Chart) at 7-12 (“For example, Migrate to Containers automatically generates a

1 *container image*, a *Dockerfile* . . . *Kubernetes deployment* . . .”); Ex. B at Exhibit 4 (HPE ’058
 2 Infringement Chart) at 5-7 (“These containers are based on *Docker* ... The idea of containerization is to
 3 isolate and *package the application with all the dependencies in a container.*”); Ex. D at Exhibit 4
 4 (Amazon ’058 Infringement Chart) at 3-4 (“Amazon ECS uses Docker images.”). Even more so than the
 5 ’814 patent, VirtaMove’s ’058 infringement contentions universally – for all Texas Cases Accused
 6 Products – identify the use of Dockerfiles, Docker images, and Docker containers. Indeed, VirtaMove’s
 7 accusations further suggest that packaging an application with its dependencies is all that is required to
 8 infringe the limitations of the ’058 patent. Based on Red Hat’s understanding of VirtaMove’s contentions,
 9 without conceding the scope or functionality of the Asserted Patents, Red Hat OpenShift uses the same
 10 open source technology that VirtaMove accuses of infringement.

11 59. Despite not naming Red Hat’s OpenShift as an accused product in any of the above cases,
 12 VirtaMove’s infringement allegations—focused on the use and deployment of Kubernetes, Docker, and
 13 other open source software programs—against the same containerization and container orchestration
 14 technologies used in OpenShift create a justiciable controversy between Red Hat and VirtaMove. Indeed,
 15 VirtaMove has initiated litigation against IBM, HPE, Amazon, and Google alleging that the same
 16 containerization and container orchestration technology—*e.g.*, use of containers and Kubernetes to build,
 17 manage, and deploy containerized applications in network infrastructures—infringes two VirtaMove
 18 patents. Red Hat OpenShift allows for building container base images comprising an application and its
 19 dependencies from Dockerfiles, deploying those images into servers to run as containers, and using
 20 Kubernetes as a container orchestration platform. Based on the accusations against the existing
 21 Defendants, which are directed at technology used in Red Hat’s OpenShift, it is not a matter of “if”
 22 VirtaMove will bring an action against Red Hat accusing OpenShift of infringement, but “when.”

23 *VIRTAMOVE’S MOST RECENT LAWSUITS CONFIRM ITS INTENT TO ENFORCE THE ASSERTED*
 24 *PATENTS AGAINST RED HAT’S OPENSIFT*

25 60. On December 20, 2024—a little over a week after the Court’s hearing on VirtaMove’s motion
 26 to dismiss—VirtaMove extended its litigation campaign against the same containerization and container
 27 orchestration technologies used in Red Hat’s OpenShift by asserting the same ’814 and ’058 Patents
 28 against Microsoft Corporation (“Microsoft”) and Oracle Corporation (“Oracle”) in the Western District

1 of Texas. *See* Ex. M (*VirtaMove, Corp. v. Microsoft Corp.*, Case No. 7:24-cv-00338 (W.D. Tex. Dec. 20,
 2 2024)); Ex. N (*VirtaMove, Corp. v. Oracle Corp.*, Case No. 7:24-cv-00338 (W.D. Tex. Dec. 20, 2024)).
 3 As with VirtaMove’s infringement contentions against IBM, HPE, Google, and Amazon, VirtaMove’s
 4 newest infringement contentions against Microsoft and Oracle focus on the same use and deployment of
 5 Kubernetes, Docker, and other open source software programs and could have just as easily been levied
 6 against Red Hat OpenShift without altering any of their substance. Moreover, the “Accused
 7 Instrumentalities” identified in VirtaMove’s recently-filed contentions encompass Microsoft and Oracle’s
 8 distribution and/or use of RedHat’s OpenShift itself, further confirming that Red Hat and its OpenShift
 9 product are squarely in VirtaMove’s crosshairs.

10 61. First, based on Red Hat’s understanding of VirtaMove’s contentions (and without conceding
 11 the scope or functionality of the Asserted Patents), VirtaMove’s newest lawsuits against Microsoft and
 12 Oracle target the sale and use of the same application containerization and container orchestration
 13 technologies used in Red Hat’s OpenShift. For example, VirtaMove’s infringement contentions against
 14 Microsoft and Oracle identify the use of Docker containers—specifically, the use of Dockerfiles and
 15 Docker images to create Docker containers in a Kubernetes environment—as infringing the Asserted
 16 Patents. *See, e.g.*, Ex. M (Microsoft Compl. Ex. 2) at 7-9, 14-16, 24-28, 36-37, 41-43, 46-48, 52, 55-57,
 17 60-64; Ex. M (Microsoft Compl. Ex. 4) at 7-18, 22-27, 33-35, 40-43, 47; Ex. N (Oracle Compl. Ex. 2) at
 18 6-18, 25-34, 36-48; Ex. N (Oracle Compl. Ex. 4) at 6-20, 28-31, 38-40, 44. Indeed, the infringement charts
 19 attached to VirtaMove’s complaints against Microsoft and Oracle rely on figures and text about
 20 Kubernetes and Docker containerization that are identical to the figures and text VirtaMove relies on in
 21 its infringement contentions against IBM, Amazon, Google, and HPE. *See, e.g.*, Ex. M (Microsoft Compl.
 22 Ex. 2) at 24-34, 55-57, 62-71; Ex. M (Microsoft Compl. Ex. 4) at 8, 10-20, 23-31, 40, 47; Ex. N (Oracle
 23 Compl. Ex. 2) at 9, 14-24, 35, 39-41, 45-54; Ex. N (Oracle Compl. Ex. 4) at 13, 16-37, 42-44. While Red
 24 Hat disputes that these features (and all of Red Hat’s features) infringe VirtaMove’s patents, the features
 25 accused by VirtaMove for each element in its infringement charts are likewise offered in and supported
 26 by Red Hat OpenShift. As in its complaints against IBM, Amazon, Google, and HPE, VirtaMove’s
 27 infringement contentions against Microsoft and Oracle confirm that VirtaMove is accusing the
 28

1 containerization and container orchestration features provided by Kubernetes and Docker, without regard
2 or dependence on a particular implementation of those features by either company.

3 62. Second, in addition to accusing same Kubernetes and Docker containerization technology
4 used in Red Hat's OpenShift, based on Red Hat's understanding of VirtaMove's contentions, VirtaMove
5 accuses Microsoft and Oracle's respective sale and/or use of Red Hat's OpenShift product itself as
6 infringing the Asserted Patents.

7 63. In its complaint against Microsoft, VirtaMove identifies the "Accused Instrumentalities" as
8 "Microsoft products and services *using secure containerized applications*, including *without limitation*
9 Azure Kubernetes Service ("AKS"), Azure Arc-enabled Kubernetes, Azure Container Registry, and Azure
10 Container Apps, and *all versions and variations thereof* since the issuance of the asserted patent." Ex. M
11 (Microsoft Compl. Ex. 2) at 1 (emphasis added); *see also* Ex. M (Microsoft Compl. Ex. 4) at 1 (identifying
12 "Microsoft products and services using user mode critical system elements as shared libraries including
13 without limitation Azure Kubernetes Service ("AKS"), Azure Arc-enabled Kubernetes, Azure Container
14 Registry, and Azure Container Apps, and all versions and variations thereof since the issuance of the
15 asserted patent.").

16 64. Based on VirtaMove's allegations, and without conceding the scope or functionality of the
17 Asserted Patents, one Microsoft service that "us[es] secure containerized applications" is "Azure Red Hat
18 OpenShift" ("ARO"). Ex. O (Barrett Decl.) ¶ 3. Azure Red Hat OpenShift is one of Microsoft's other
19 application containerization services like Azure Kubernetes Service ("AKS"), providing the Kubernetes-
20 based OpenShift platform (along with its application containerization and Docker management features
21 accused by VirtaMove) to customers through Microsoft's Azure system. Ex. O (Barrett Decl.) ¶ 3. Azure
22 Red Hat OpenShift is "jointly engineered, operated, and supported by Red Hat and Microsoft to provide
23 an integrated support experience," further providing "highly available, fully managed OpenShift clusters
24 on demand, monitored and operated jointly by Microsoft and Red Hat." Ex. P
25 (<https://learn.microsoft.com/en-us/azure/openshift/intro-openshift>); Ex. Q
26 (<https://azure.microsoft.com/en-us/products/openshift>); Ex. O (Barrett Decl.) ¶ 3. And just like the
27 OpenShift product that Red Hat offers directly to its customers, "Kubernetes is at the core of [the] Red
28

1 Hat OpenShift” offering provided by Microsoft. Ex. Q ([https://azure.microsoft.com/en-](https://azure.microsoft.com/en-us/products/openshift)
2 [us/products/openshift](https://azure.microsoft.com/en-us/products/openshift)).

3 65. VirtaMove’s complaint against Microsoft does nothing to exclude OpenShift from its
4 identified “Accused Instrumentalities,” that according to VirtaMove include, “without limitation,” any
5 “Microsoft products and services using secure containerized applications.” Ex. M (Microsoft Compl. Ex.
6 2) at 1. On information and belief, VirtaMove is aware that the Azure Red Hat OpenShift is a “Microsoft
7 product[] and service[] using secure containerized applications” and intends for its contentions to cover
8 Azure Red Hat OpenShift, as those contentions cite to Microsoft webpages that identify Azure Red Hat
9 OpenShift as one of the managed containerization solutions made available in Azure. *Compare* Ex. M
10 (Microsoft Compl. Ex. 2) at 1 *and* Ex. M (Microsoft Compl. Ex. 4) at 1 (citing
11 <https://learn.microsoft.com/en-us/azure/aks/what-is-aks>) *with* Ex. R ([https://learn.microsoft.com/en-](https://learn.microsoft.com/en-us/azure/aks/what-is-aks)
12 [us/azure/aks/what-is-aks](https://learn.microsoft.com/en-us/azure/aks/what-is-aks)) (listing “Azure Red Hat OpenShift” as one of the “managed Kubernetes”
13 container solutions offered by Microsoft). VirtaMove’s affirmative decision to identify the “Accused
14 Instrumentalities” in a way that encompasses Microsoft’s use and distribution of OpenShift, and its refusal
15 to limit such allegations to only Microsoft’s *non*-OpenShift offerings, further confirms the substantial and
16 immediate risk of a suit against Red Hat and further establishes the justiciable controversy between Red
17 Hat and VirtaMove.

18 66. In its complaint against Oracle, VirtaMove identifies the “Accused Instrumentalities” as
19 “Oracle products and services using secure containerized applications, including without limitation Oracle
20 Cloud Infrastructure (“OCI”) and Oracle Kubernetes Engine (“OCE”) [sic], and all versions and variations
21 thereof since the issuance of the asserted patent.” Ex. N (Oracle Compl. Ex. 2) at 1; *see also* Ex. N (Oracle
22 Compl. Ex. 4) at 1 (identifying “Oracle products and services using user mode critical system elements as
23 shared libraries, including without limitation Oracle Cloud Infrastructure (“OCI”) and Oracle Kubernetes
24 Engine (“OCE”) [sic], and all versions and variations thereof since the issuance of the asserted patent.”).

25 67. The Oracle Cloud Infrastructure (“OCI”) identified by VirtaMove as an “Accused
26 Instrumentality” allows customers to deploy and run “Red Hat’s OpenShift container platform” on
27 Oracle’s OCI to create, support, and manage containerized applications (including Docker containers)
28 using OpenShift’s Kubernetes features. Ex. S (<https://www.oracle.com/news/announcement/red-hat->

1 [openshift-generally-available-on-oracle-cloud-infrastructure-2024-05-06/](https://docs.oracle.com/en-us/iaas/Content/openshift-on-oci/overview.htm); Ex. T
 2 (<https://docs.oracle.com/en-us/iaas/Content/openshift-on-oci/overview.htm>) (“Red Hat OpenShift
 3 Container Platform is a cloud-based Kubernetes container platform. Red Hat, in partnership with OCI,
 4 supports running cluster workloads on the OCI platform.”); Ex. O (Barrett ¶ 4). This offering was made
 5 generally available in May 2024 as part of a “partnership” with Red Hat and is a variation of Oracle’s
 6 other containerization solution, “Oracle Kubernetes Engine,” which VirtaMove also identifies as an
 7 “Accused Instrumentalit[y].” Ex. M (Microsoft Compl. Ex. 2) at 1; *see also* Ex. M (Microsoft Compl.
 8 Ex. 4) at 1. Because VirtaMove does not exclude the use of Red Hat’s OpenShift on Oracle’s OCI platform
 9 from the scope of its infringement allegations against Oracle, Red Hat understands that VirtaMove
 10 includes the use of OpenShift with the scope of its infringement allegations against Oracle.

11 68. VirtaMove’s recent affirmative conduct described above demonstrates an intent to enforce the
 12 Asserted Patents against Red Hat’s OpenShift product. In its lawsuits against IBM, Amazon, Google, and
 13 HPE, VirtaMove accused the same Kubernetes and Docker containerization technology used in Red Hat’s
 14 OpenShift. In its most recent lawsuits against Microsoft and Oracle, VirtaMove again incorrectly asserts
 15 infringement against that same technology and against use of Red Hat’s OpenShift itself, further
 16 establishing a cognizable case and controversy over Red Hat’s claims for declaratory judgment.

17 *RED HAT DOES NOT INFRINGE THE ASSERTED PATENTS*

18 69. Red Hat OpenShift does not directly or indirectly infringe any claim of the Asserted Patents.
 19 To the best of Red Hat’s knowledge, no third party infringes any claim of the Asserted Patents by using
 20 Red Hat OpenShift. Red Hat has not caused, directed, requested, or facilitated any such infringement,
 21 must less with the specific intent to do so. Red Hat OpenShift is not designed for use in any combination
 22 which infringes any claim of the Asserted Patents. To the contrary, it is a product with substantial uses
 23 that do not infringe any claim of the Asserted Patents.

24 70. Among other reasons, Red Hat OpenShift does not infringe the ’814 patent because each of
 25 the containers in the product do not have a “unique root file system that is different from an operating
 26 system’s root file system,” as required by the claims of the ’814 patent. Moreover, Red Hat OpenShift
 27 does not infringe the ’814 patent because it does not practice “each container being mutually exclusive of
 28

1 the other, such that read/write files within a container cannot be shared with other containers,” as required
2 by the remaining claims of the ’814 patent.

3 71. Among other reasons, Red Hat OpenShift does not infringe the ’058 patent because the
4 product does not “store in the shared library . . . functional replicas of OSCSEs,” as required by all claims
5 of the ’058 patent. Rather, at build time, the program pulls a container base image from the container
6 registry, upon which application code and application dependencies can be built. More specifically,
7 application code and application dependencies are installed into the container base image. Applications
8 are not accessing a shared library for application dependencies and becoming distinct operating system
9 environments thereafter.

10 **FIRST COUNT**
(Declaration of Non-Infringement of U.S. Patent No. 7,519,814)

11 72. Red Hat restates and incorporates by reference all allegations in this Complaint as if fully set
12 forth herein.

13 73. VirtaMove claims to own all rights, title, and interest, including the right to seek damages for
14 past, present, and future infringement thereof, in the ’814 patent. A true and correct copy of the ’814
15 patent is attached hereto as Exhibit J.

16 74. In the case VirtaMove has brought against IBM, VirtaMove accuses IBM of infringing the
17 ’814 patent based on allegations that focus solely on functionality provided by Kubernetes, Docker, and
18 other open source software—functionalities integrated into Red Hat OpenShift.

19 75. A substantial, immediate, and real controversy exists between Red Hat and VirtaMove
20 regarding whether Red Hat OpenShift infringes or has infringed the ’814 patent. A judicial declaration is
21 necessary to determine the parties’ respective rights regarding the ’814 patent.

22 76. Red Hat seeks a judgment declaring that Red Hat OpenShift does not directly or indirectly
23 infringe any claim of the ’814 patent. In its complaint, and infringement contentions, against IBM,
24 VirtaMove cites to IBM Kubernetes as purported evidence of infringement of claims 1, 2, 6, 9, and 10 of
25 the ’814 patent. Based on Red Hat’s present understanding of claims 1, 2, 6, 9, and 10 of the ’814 patent
26 and VirtaMove’s allegations, Red Hat OpenShift fails to meet or embody the limitations of claims 1, 2, 6,
27 9, and 10 of the ’814 patent.

SECOND COUNT
(Declaration of Non-Infringement of U.S. Patent No. 7,784,058)

77. Red Hat restates and incorporates by reference all allegations in this Complaint as if fully set forth herein.

78. VirtaMove claims to own all rights, title, and interest, including the right to seek damages for past, present, and future infringement thereof, in the '058 patent. A true and correct copy of the '058 patent is attached hereto as Exhibit K.

79. In the case VirtaMove has brought against IBM, VirtaMove accuses IBM of infringing the '058 patent based on allegations that focus solely on functionality provided by Kubernetes, Docker, and other open source software—functionalities integrated into Red Hat OpenShift.

80. A substantial, immediate, and real controversy exists between Red Hat and VirtaMove regarding whether Red Hat OpenShift infringes or has infringed the '058 patent. A judicial declaration is necessary to determine the parties' respective rights regarding the '058 patent.

81. Red Hat seeks a judgment declaring that Red Hat OpenShift does not directly or indirectly infringe any claim of the '058 patent. In its complaint, and infringement contentions, against IBM, VirtaMove cites to IBM Kubernetes as purported evidence of infringement of claims 1, 2, 3, 4, and 18 of the '058 patent. Based on Red Hat's present understanding of claims 1, 2, 3, 4, and 18 of the '058 patent and VirtaMove's allegations, Red Hat OpenShift fails to meet or embody the limitations of claims 1, 2, 3, 4, and 18 of the '058 patent.

PRAYER FOR RELIEF

WHEREFORE, Red Hat prays for judgment and relief as follows:

A. Declaring that Red Hat's products, including OpenShift, do not infringe directly or indirectly any claim of the '814 patent and enjoining VirtaMove, its officers, agents, employees, attorneys, and all persons in active concert or participation with them, from directly or indirectly charging infringement, or instituting further action for infringement, of the '814 patent against Red Hat or any of its customers;

B. Declaring that Red Hat's products, including OpenShift, do not infringe directly or indirectly any claim of the '058 patent and enjoining VirtaMove, its officers, agents, employees, attorneys, and all

persons in active concert or participation with them, from directly or indirectly charging infringement, or instituting further action for infringement, of the '058 patent against Red Hat or any of its customers;

C. Finding that this is an exceptional case under 35 U.S.C. § 285;

D. Awarding Red Hat its costs and attorneys' fees in connection with this action; and

E. Such further and additional relief as the Court deems just and proper.

JURY DEMAND

Red Hat demands a jury trial on all issues and claims so triable.

DATED: January 13, 2025

Respectfully submitted,

KIRKLAND & ELLIS LLP

/s/ Todd M. Friedman

Brandon H. Brown (SBN 266347)

Kyle Calhoun (SBN 311181)

KIRKLAND & ELLIS LLP

555 California Street

San Francisco, CA 94104

Telephone: (415) 439-1400

Facsimile: (415) 439-1500

Email: brandon.brown@kirkland.com

Email: kyle.calhoun@kirkland.com

Todd M. Friedman (admitted *pro hac vice*)

KIRKLAND & ELLIS LLP

601 Lexington Avenue

New York, NY 10022

Telephone: (212) 446-4800

Facsimile: (212) 446-4900

Email: todd.friedman@kirkland.com

Attorneys for Plaintiff RED HAT, INC.

EXHIBIT A

**UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

<p>VIRTAMOVE, CORP.,</p> <p style="text-align: center;">Plaintiff,</p> <p style="text-align: center;">v.</p> <p>HEWLETT PACKARD ENTERPRISE COMPANY,</p> <p style="text-align: center;">Defendant.</p>	<p>Case No. 2:24-cv-00093-JRG</p> <p style="text-align: center;">(LEAD CASE)</p> <p>JURY TRIAL DEMANDED</p>
<p>VIRTAMOVE, CORP.,</p> <p style="text-align: center;">Plaintiff,</p> <p style="text-align: center;">v.</p> <p>INTERNATIONAL BUSINESS MACHINES CORP.,</p> <p style="text-align: center;">Defendant.</p>	<p>Case No. 2:24-CV-00064-JRG</p> <p style="text-align: center;">(Member case)</p> <p>JURY TRIAL DEMANDED</p>

**SECOND AMENDED COMPLAINT FOR PATENT INFRINGEMENT AGAINST
INTERNATIONAL BUSINESS MACHINES CORP.**

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, “Plaintiff” or “VirtaMove”) makes the following allegations against Defendant International Business Machines Corp. (collectively, “Defendant” or “IBM”):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant’s unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058

(collectively, the “Asserted Patents”). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as Appzero Software Corp. (“Appzero”), which was established in 2010.

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repackage, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove’s mission is to move and modernize the world’s server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach \$2.1 billion in 2019 and \$4.3 billion in 2022—a compound annual growth rate (“CAGR”) of 30%. *See, e.g.,* <https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022>. Another source reported the application containers market had a market size of \$5.45 billion in

2024 and estimated it to reach \$19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.*, <https://www.mordorintelligence.com/industry-reports/application-container-market>.

6. Defendant International Business Machines Corp. is a New York corporation. IBM has a principal place of business at One New Orchard Road, Armonk, New York 10504. IBM may also be served with process via its registered agent Amanda Garcia, at 330 North Brand Blvd, Glendale, California 91203.

JURISDICTION AND VENUE

7. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

9. Venue is proper in this District. For example, IBM has a regular and established place of business, including, e.g., at 1700 Summit Avenue, Plano, Texas 75074.

ADDITIONAL FACTS

10. It is reported by 451 Research that:

Since September 2011, IBM Distinguished Engineer Mac Devine has served as the Director and CTO for the SmartCloud portfolio within IBM Global Technology Services (GTS). In practice, that

makes him responsible for SmartCloud's technical strategy and architecture, and for choosing SmartCloud partners. Most recently, Devine has been involved in a partnership between AppZero and IBM to provide application migration services to enterprises as they adopt cloud computing.

Ex. 5 at 1. It is further reported that:

It was Devine who performed the due diligence exercise in 2011 that reviewed IBM's options for migration tools. He narrowed down the options to CohesiveFT and AppZero, and then he introduced the companies to each other. Through the resulting Elastic Enterprise Applications partnership, Cohesive handles networking and automation for Linux, while AppZero offers a Windows virtualized application container. Devine is proud that the partners got an offering up and operational in six weeks.

Id. at 3.

11. Additionally, representatives of IBM met with representatives of VirtaMove in 2016, 2017, 2018, and 2021 to discuss and demo AppZero and its technology for either partnership, use, distribution, and/or investment. However, IBM appears to have copied or reverse engineered the AppZero technology. Either IBM would have known, as a result of its "due diligence" and ongoing discussions, that AppZero was patented as early as 2011, or IBM intentionally stayed willfully blind to this fact.

COUNT I

INFRINGEMENT OF U.S. PATENT NO. 7,519,814

12. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

13. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 ('814 patent), titled "System for Containerization of Application Sets," issued on April 14, 2009. A true

and correct copy of the '814 Patent is attached as Exhibit 1.

14. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., IBM's IBM Cloud Kubernetes Service, that directly infringe, literally and/or under the doctrine of equivalents, claims of the '814 patent, for example:

Run Kubernetes at enterprise scale

Experience a certified, managed Kubernetes solution, built for creating a cluster of compute hosts to deploy and manage containerized apps on IBM Cloud®. IBM manages the master, freeing you from having to administer the host OS, container runtime and Kubernetes version-update process.

<https://www.ibm.com/products/kubernetes-service>.

15. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

16. Defendant's infringement has been and is willful. Defendant knew of VirtaMove, its products, and at least one of the patents long before this suit was filed and at least as early as 2012. For example, on or about February 2012, in connection with the prosecution of U.S. Patent App. No. 12/146,322 (assigned to IBM), the examiner cited U.S. Pub. No. 2005/0060722 (which issued as the '814 Patent) against IBM. On or about November 26, 2012, in connection with the prosecution of U.S. Patent No. 8,893,306 (assigned to IBM), the examiner cited the '814 Patent

against IBM. On or about June 2015, in connection with the prosecution of U.S. Patent No. 9,166,865 (assigned to IBM), the examiner cited U.S. Pub. No. 2005/0060722 against IBM. Defendant knew, or should have known, that its conduct amounted to infringement of the '814 patent. Accordingly, Defendant is liable for willful infringement.

17. Defendant also knowingly and intentionally induces infringement of claims of the '814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and/or earlier, as set forth above. Despite this knowledge of the '814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '814 patent. Defendant does so, knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '814 patent, thereby specifically intending for and inducing its customers to infringe the '814 patent through the customers' normal and customary use of the Accused Products.

18. Defendant has also infringed, and continue to infringe, claims of the '814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '814 patent, in violation of 35 U.S.C. § 271(c).

19. The Accused Products satisfy all claim limitations of claims of the '814 patent. A claim chart comparing an independent claim of the '814 patent to a representative Accused Product, is attached as Exhibit 2, which is hereby incorporated by reference in its entirety.

20. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the '814 patent pursuant to 35 U.S.C. § 271.

21. As a result of Defendant's infringement of the '814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287.

COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

22. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

23. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 ('058 patent), titled "Computing System Having User Mode Critical System Elements as Shared Libraries," issued on August 24, 2010. A true and correct copy of the '058 patent is attached as Exhibit 3.

24. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., IBM's IBM Cloud Kubernetes Service, that directly infringe, literally and/or under the doctrine of equivalents, claims of the '058 patent, for example:

Run Kubernetes at enterprise scale

Experience a certified, managed Kubernetes solution, built for creating a cluster of compute hosts to deploy and manage containerized apps on IBM Cloud®. IBM manages the master, freeing you from having to administer the host OS, container runtime and Kubernetes version-update process.

<https://www.ibm.com/products/kubernetes-service>.

25. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

26. Defendant's infringement has been and is willful. Defendant knew of VirtaMove, its products, and at least one of the patents long before this suit was filed and at least as early as 2015. For example, U.S. Patent No. 9,176,713 (assigned to IBM) issued on November 3, 2015 and identifies the '058 Patent under "References Cited." Additionally, U.S. Patent No. 9,934,055 (assigned to IBM) issued on April 3, 2018 and identifies the '058 Patent under "References Cited." Defendant knew, or should have known, that its conduct amounted to infringement of the '058 patent. Accordingly, Defendant is liable for willful infringement.

27. Defendant also knowingly and intentionally induces infringement of claims of the '058 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed. Despite this knowledge of the '058 patent, Defendant continues to actively encourage and

instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '058 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '058 patent, thereby specifically intending for and inducing its customers to infringe the '058 patent through the customers' normal and customary use of the Accused Products.

28. Defendant has also infringed, and continue to infringe, claims of the '058 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '058 patent, in violation of 35 U.S.C. § 271(c).

29. The Accused Products satisfy all claim limitations of claims of the '058 patent. A claim chart comparing an independent claim of the '058 patent to a representative Accused Product, is attached as Exhibit 4, which is hereby incorporated by reference in its entirety.

30. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the '058 patent pursuant to 35 U.S.C. § 271.

31. As a result of Defendant's infringement of the '058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no

event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

- a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;
- b. A judgment in favor of Plaintiff that Defendant has willfully infringed the '814 and '058 patents;
- c. A permanent injunction prohibiting Defendant from further acts of infringement of each of the '814 and '058 patents;
- d. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant's infringement of each of the Asserted Patents;
- e. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;
- f. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys' fees against Defendant; and
- g. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury of any issues so triable by right.

Dated: May 29, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)

rmirzaie@raklaw.com

Marc A. Fenster (CA SBN 181067)

mfenster@raklaw.com

Neil A. Rubin (CA SBN 250761)

nrubin@raklaw.com

Amy E. Hayden (CA SBN 287026)

ahayden@raklaw.com

Jacob R. Buczko (CA SBN 269408)

jbuczko@raklaw.com

James S. Tsuei (CA SBN 285530)

jtsuei@raklaw.com

James A. Milkey (CA SBN 281283)

jmilkey@raklaw.com

Christian W. Conkle (CA SBN 306374)

cconkle@raklaw.com

Jonathan Ma (CA SBN 312773)

jma@raklaw.com

Daniel Kolko (CA SBN 341680)

dkolko@raklaw.com

RUSS AUGUST & KABAT

12424 Wilshire Boulevard, 12th Floor

Los Angeles, CA 90025

Telephone: (310) 826-7474

Qi (Peter) Tong (TX SBN 24119042)

ptong@raklaw.com

RUSS AUGUST & KABAT

4925 Greenville Ave., Suite 200

Dallas, TX 75206

Telephone: (310) 826-7474

Attorneys for Plaintiff VirtaMove, Corp.

CERTIFICATE OF SERVICE

I certify that on May 29, 2024 a true and correct copy of the foregoing document was electronically filed with the Court and served on all parties of record via the Court's CM/ECF system.

/s/ Reza Mirzaie

Reza Mirzaie

CERTIFICATE OF CONFERENCE

I certify that on May 28, 2024, I conferred with counsel for IBM, who confirmed that IBM is not opposed to the filing of this Second Amended Complaint under Fed. R. Civ. P. 15(a)(2), which provides, in part, that "a party may amend its pleading only with the opposing party's written consent"

/s/ Reza Mirzaie

Reza Mirzaie

Exhibit 1



US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,519,814 B2**
(45) **Date of Patent:** **Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

WO WO 2006/039181 A 4/2006

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248;
709/214; 719/319

(58) **Field of Classification Search** 713/167;
719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

OTHER PUBLICATIONS

Soltész, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667.*

Primary Examiner—Kambiz Zand

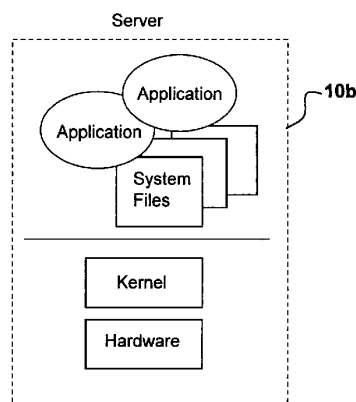
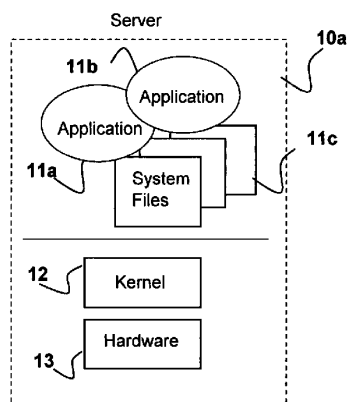
Assistant Examiner—Ronald Baum

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



US 7,519,814 B2

Page 2

U.S. PATENT DOCUMENTS				2002/0004854 A1	1/2002	Hartley	
6,847,970 B2 *	1/2005	Keller et al.	707/100	2002/0174215 A1	11/2002	Schaefer	709/224
7,076,784 B1 *	7/2006	Russell et al.	719/315	2003/0101292 A1	5/2003	Fisher	
7,287,259 B2 *	10/2007	Grier et al.	719/331	* cited by examiner			

U.S. Patent

Apr. 14, 2009

Sheet 1 of 17

US 7,519,814 B2

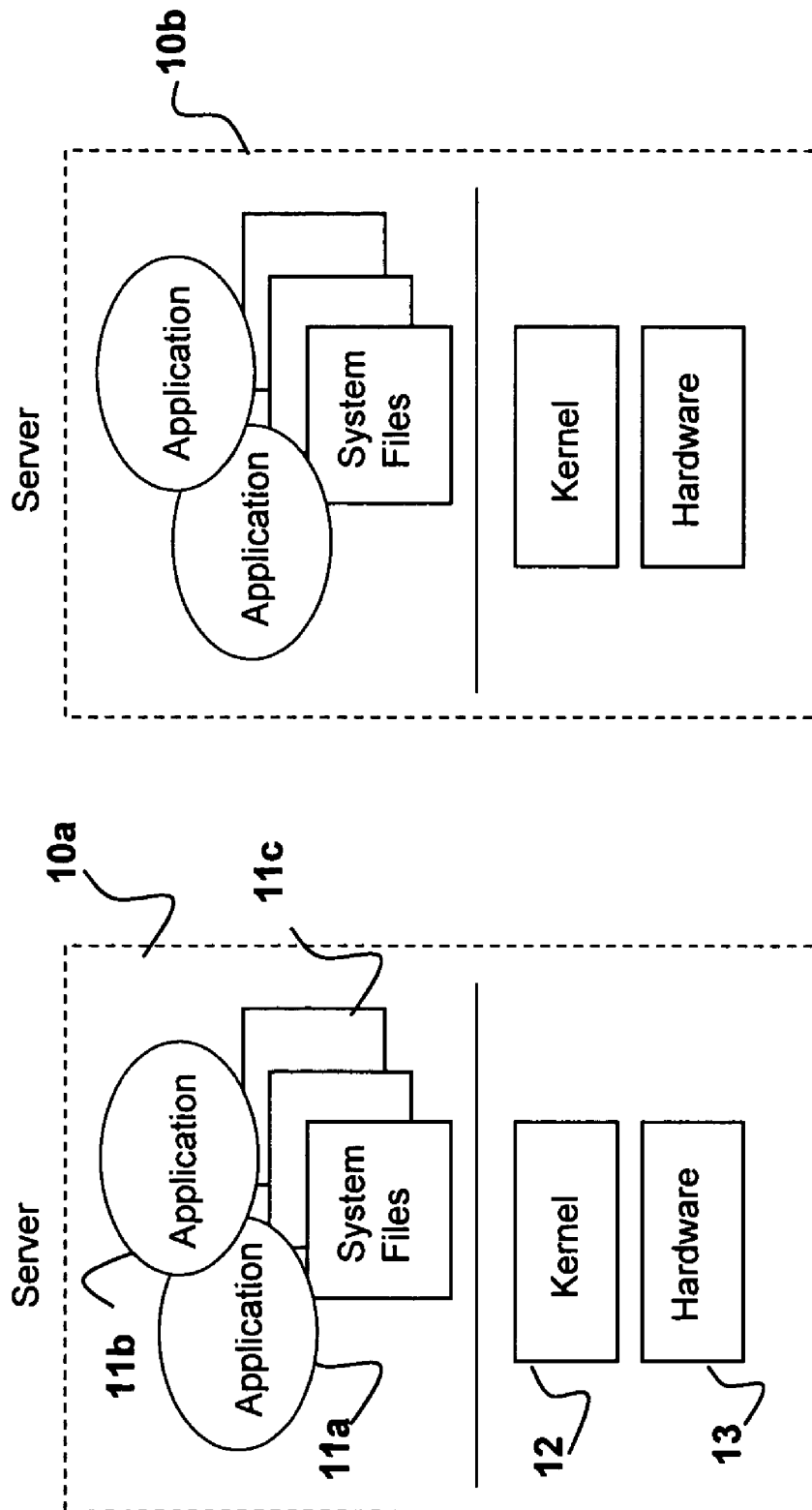


Figure 1

U.S. Patent

Apr. 14, 2009

Sheet 2 of 17

US 7,519,814 B2

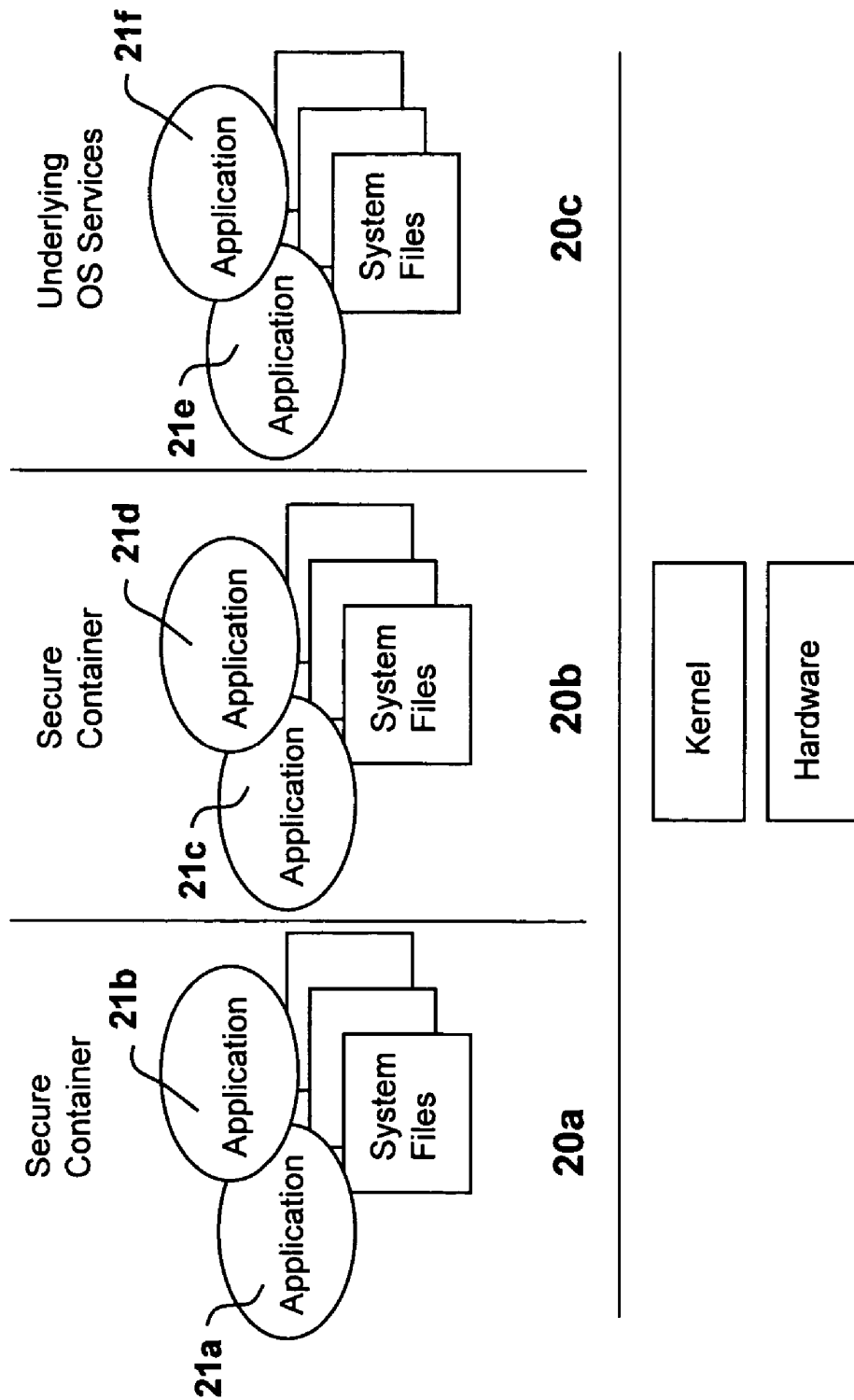


Figure 2

U.S. Patent

Apr. 14, 2009

Sheet 3 of 17

US 7,519,814 B2

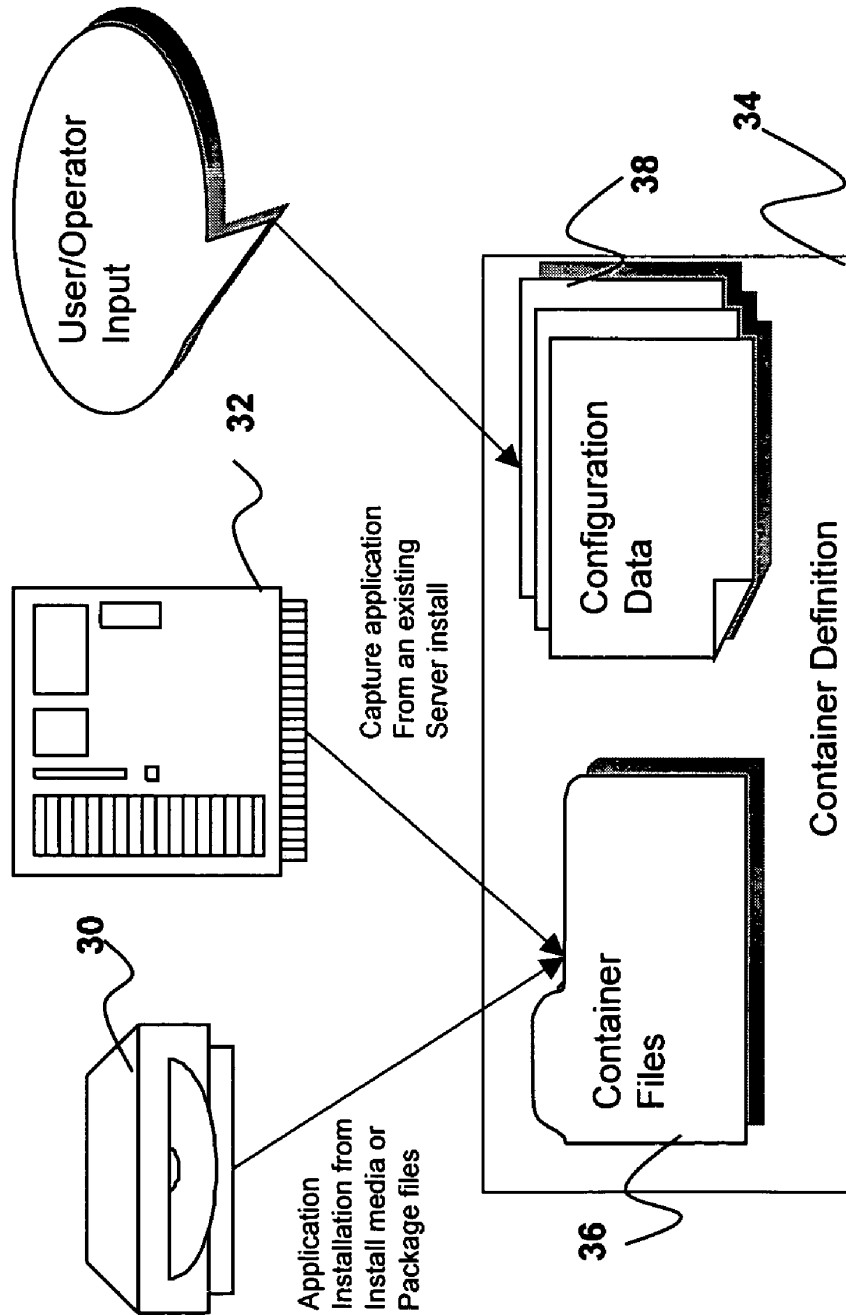


Figure 3

U.S. Patent

Apr. 14, 2009

Sheet 4 of 17

US 7,519,814 B2

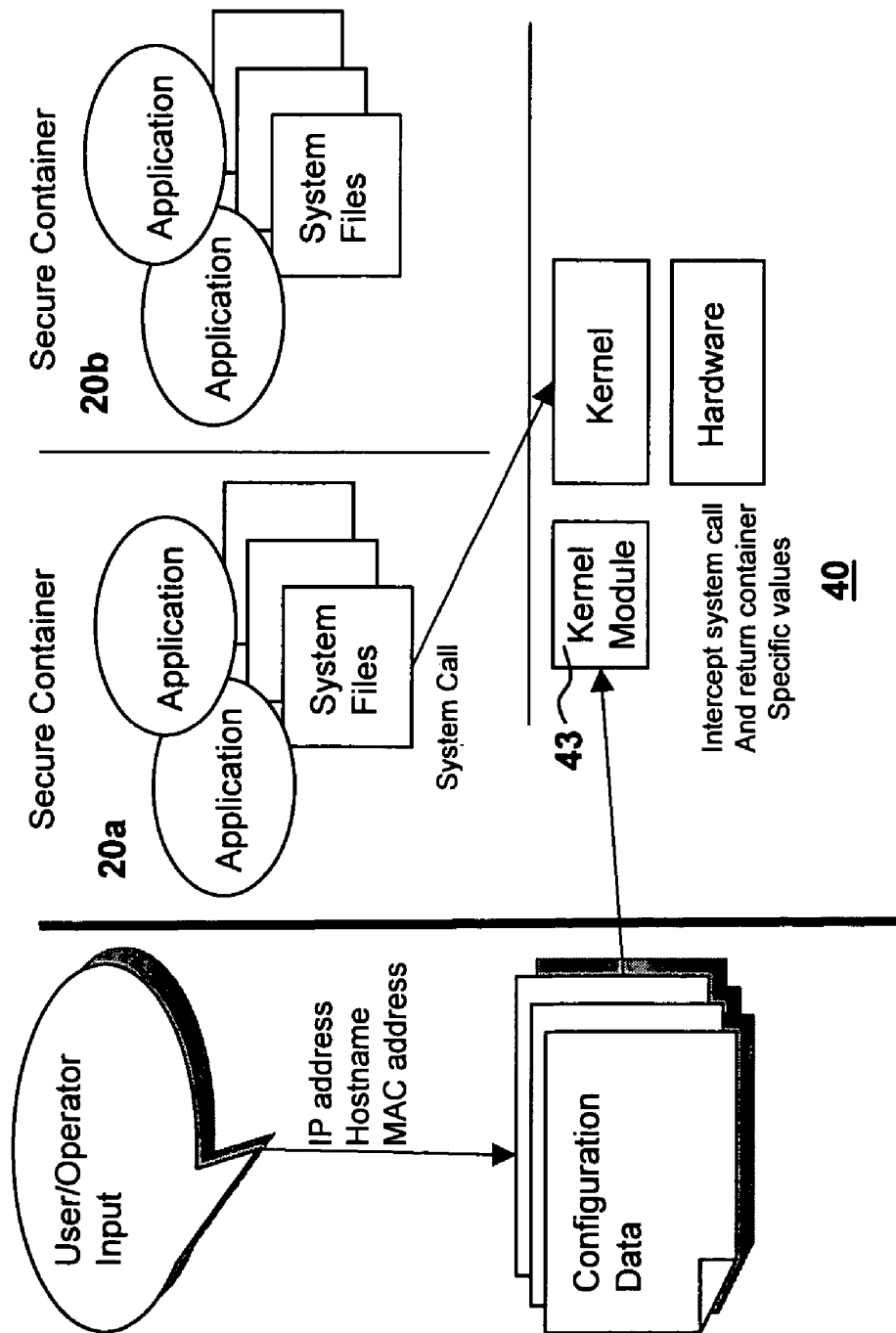


Figure 4

U.S. Patent

Apr. 14, 2009

Sheet 5 of 17

US 7,519,814 B2

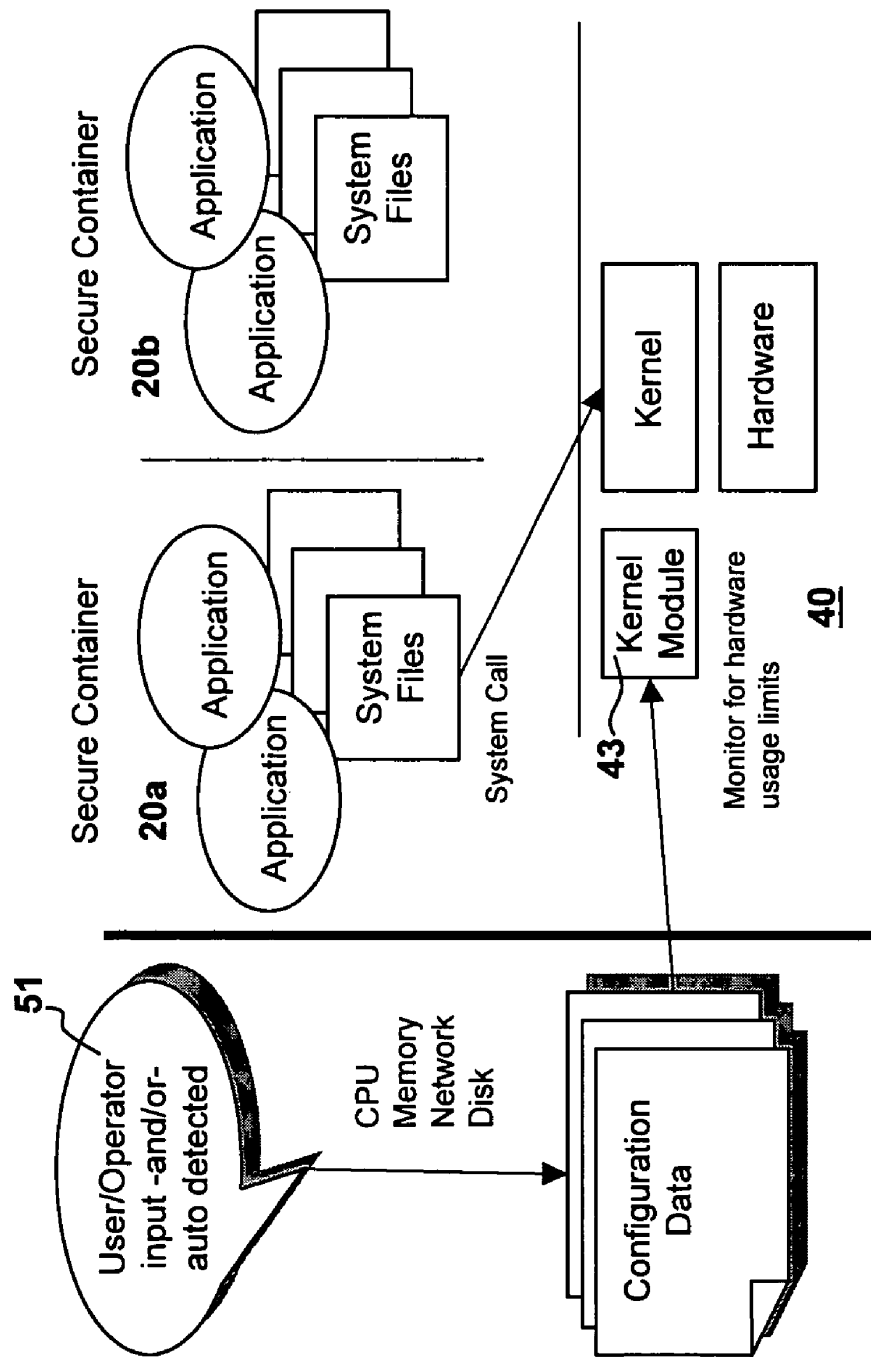


Figure 5

U.S. Patent

Apr. 14, 2009

Sheet 6 of 17

US 7,519,814 B2

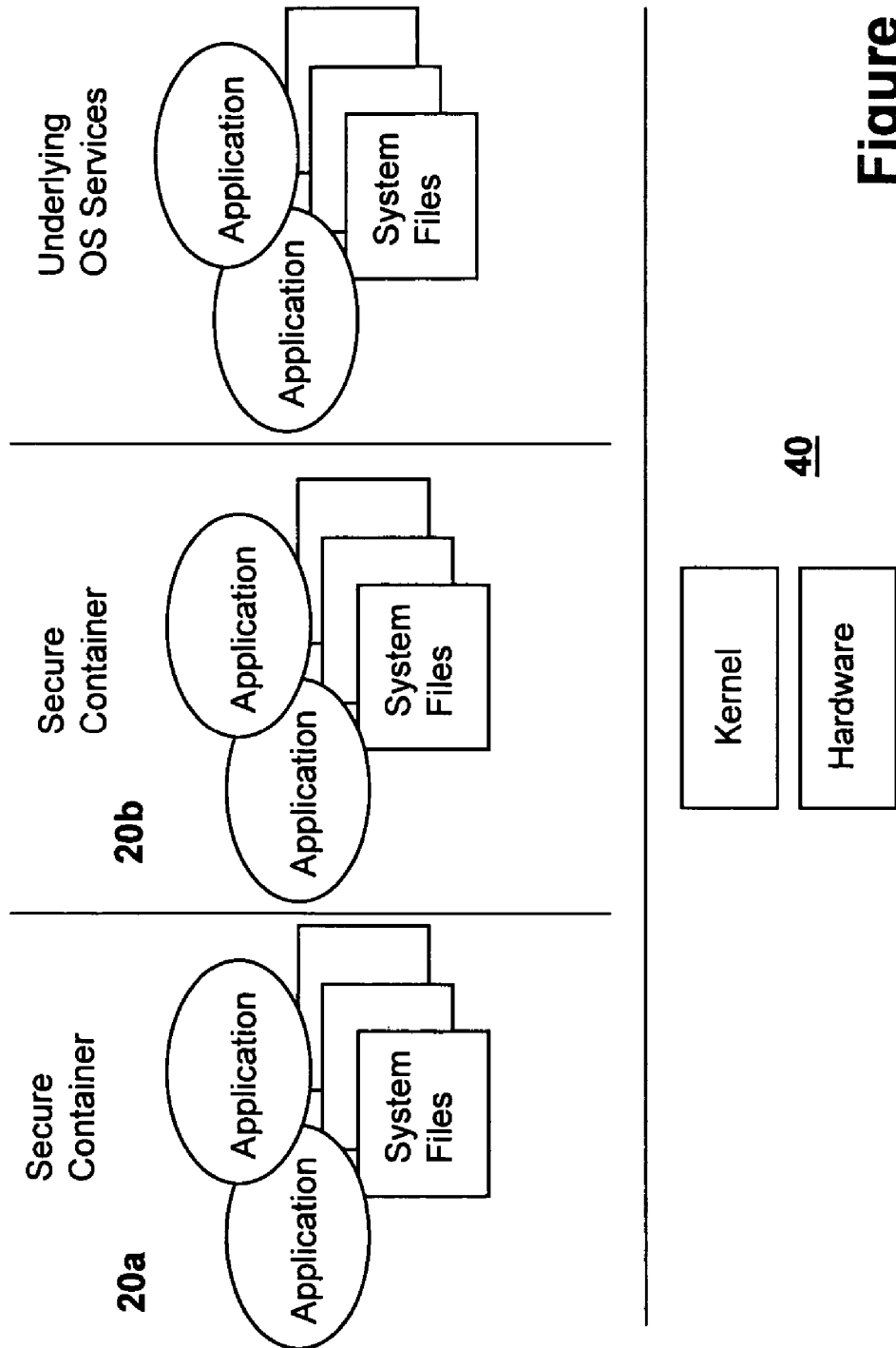


Figure 6

U.S. Patent

Apr. 14, 2009

Sheet 7 of 17

US 7,519,814 B2

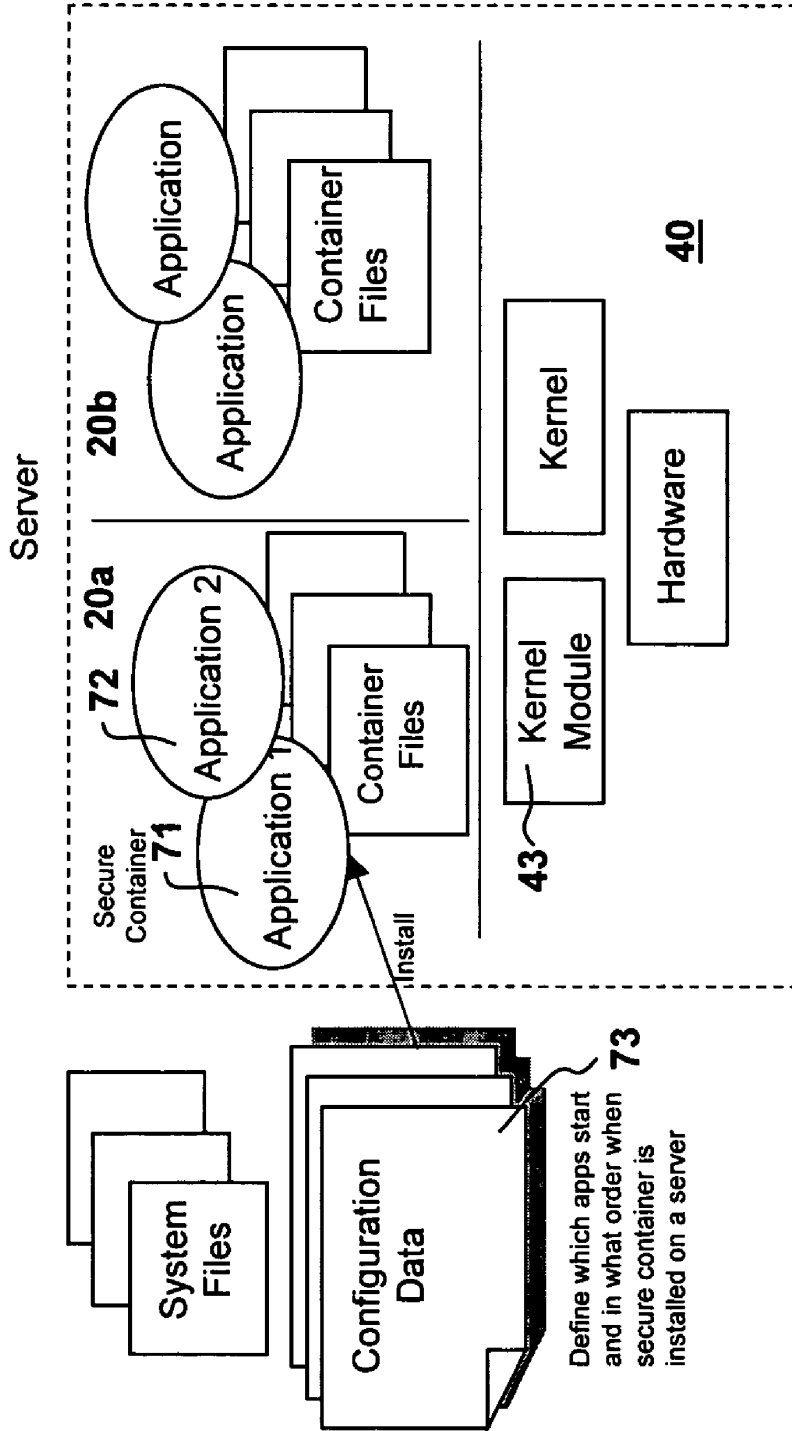


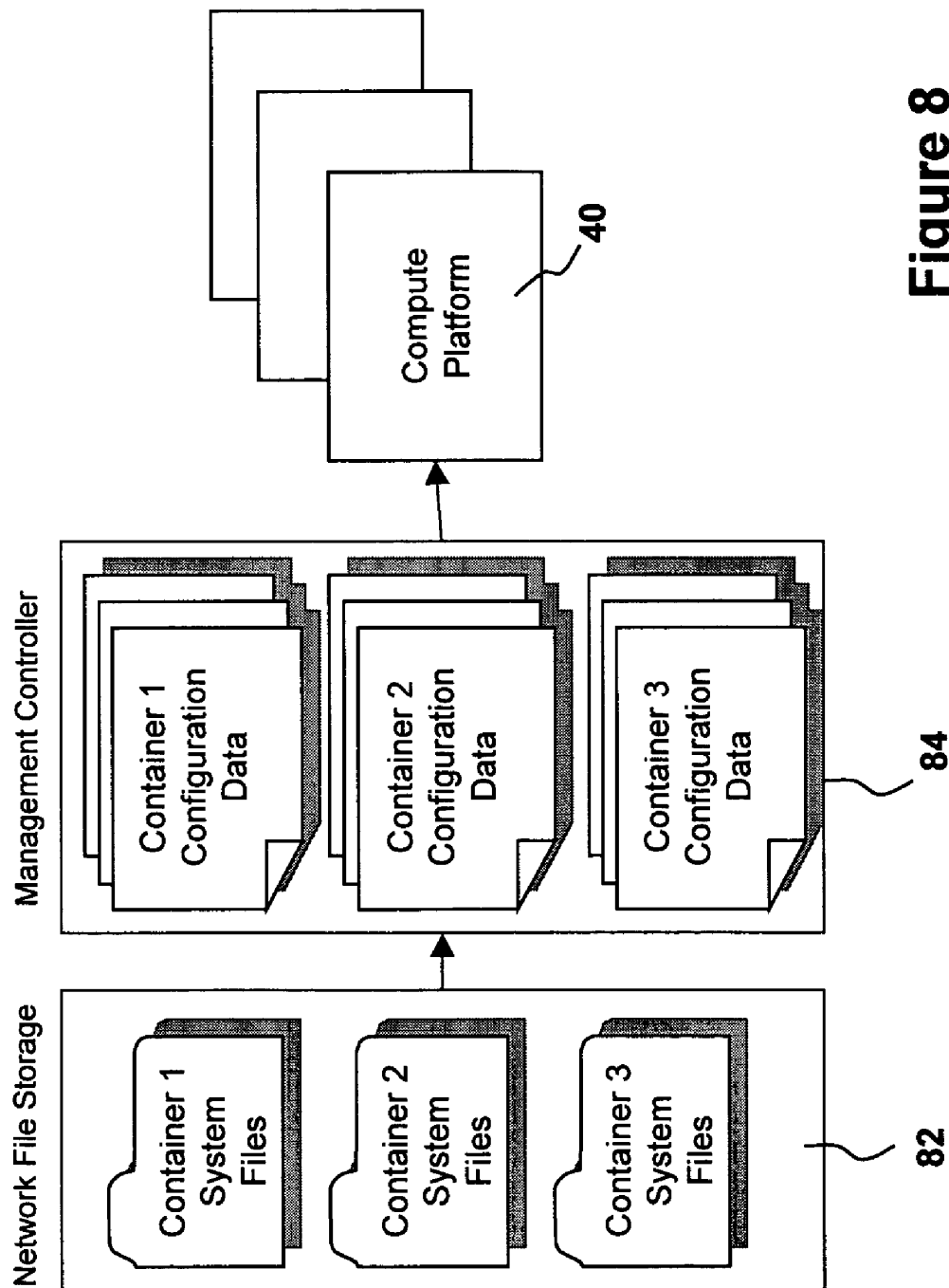
Figure 7

U.S. Patent

Apr. 14, 2009

Sheet 8 of 17

US 7,519,814 B2



U.S. Patent

Apr. 14, 2009

Sheet 9 of 17

US 7,519,814 B2

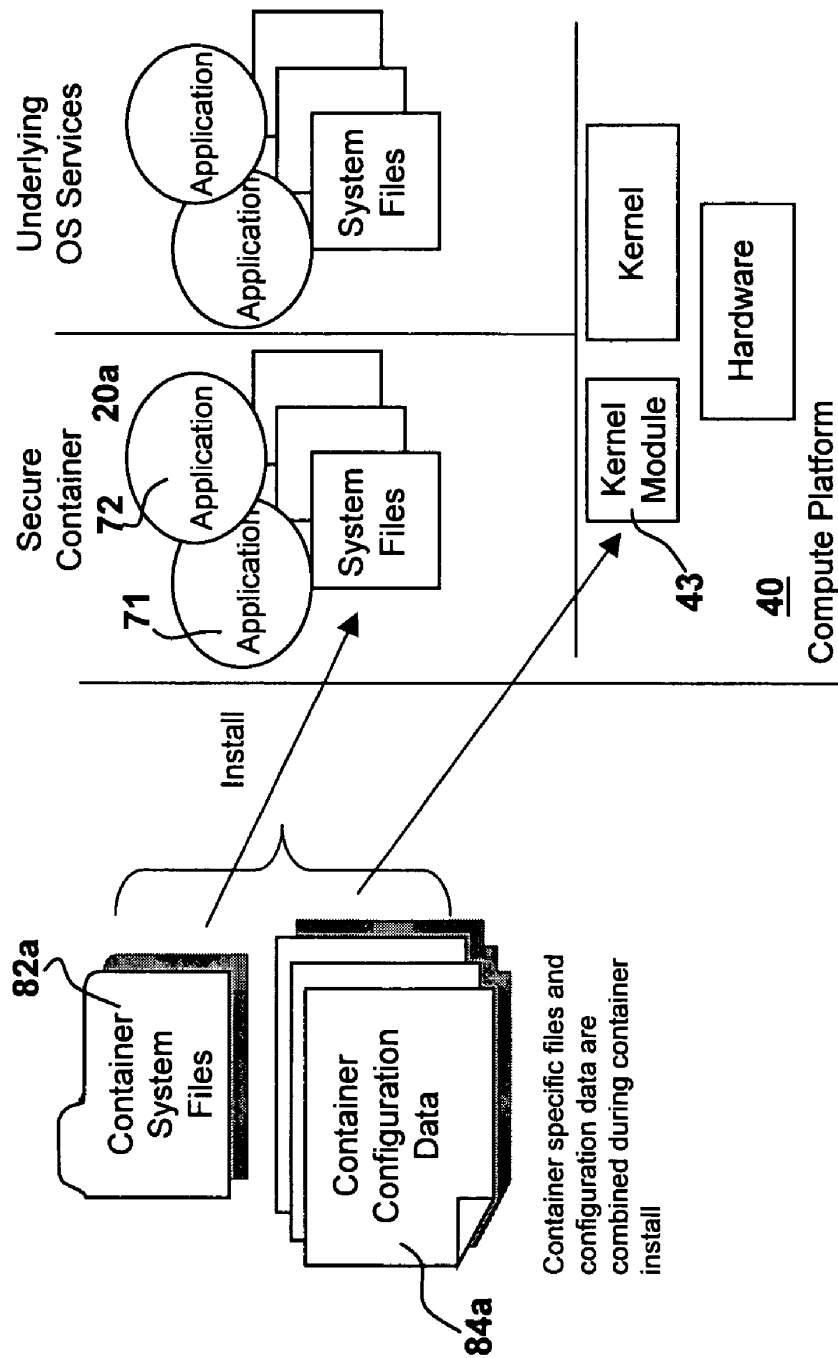


Figure 9

U.S. Patent

Apr. 14, 2009

Sheet 10 of 17

US 7,519,814 B2

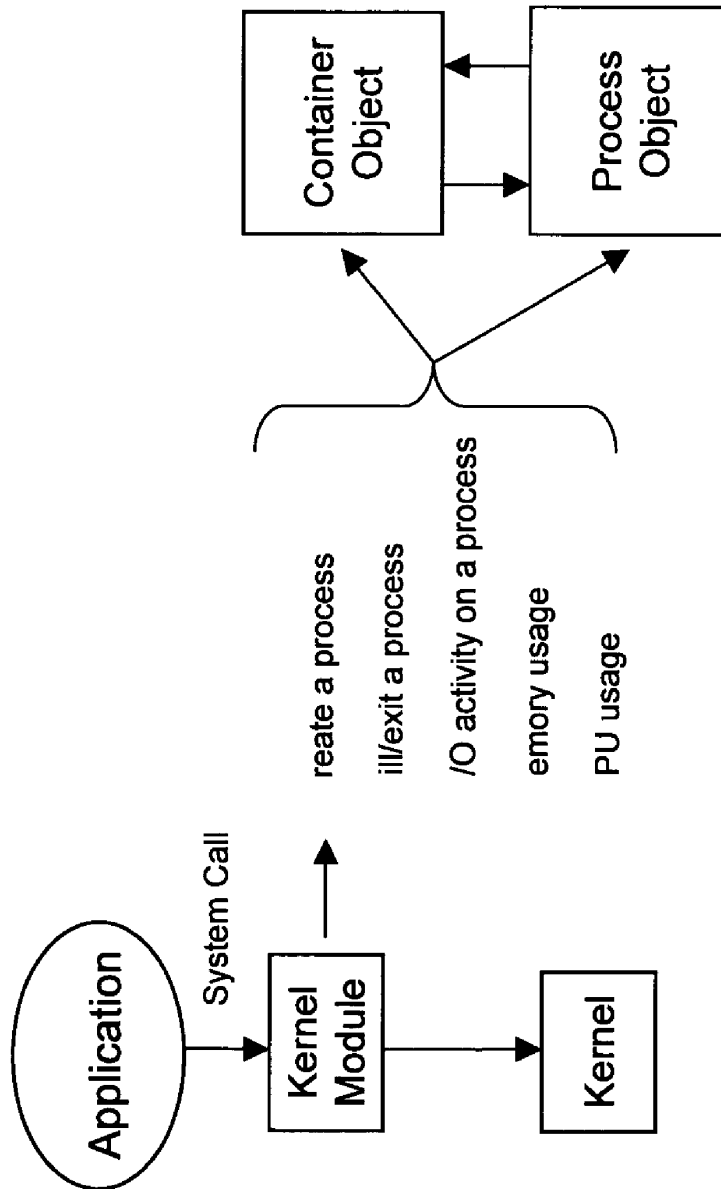


Figure 10

U.S. Patent

Apr. 14, 2009

Sheet 11 of 17

US 7,519,814 B2

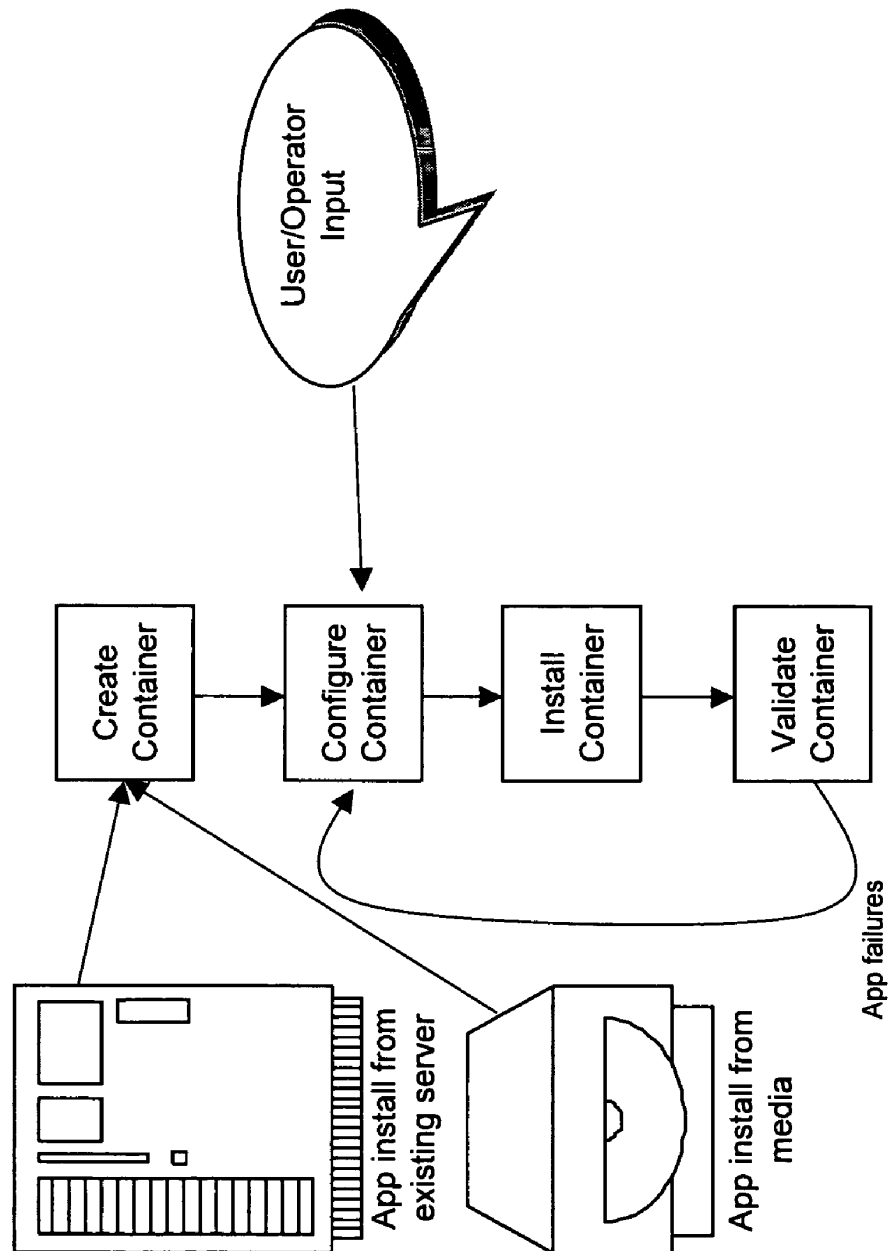


Figure 11

U.S. Patent

Apr. 14, 2009

Sheet 12 of 17

US 7,519,814 B2

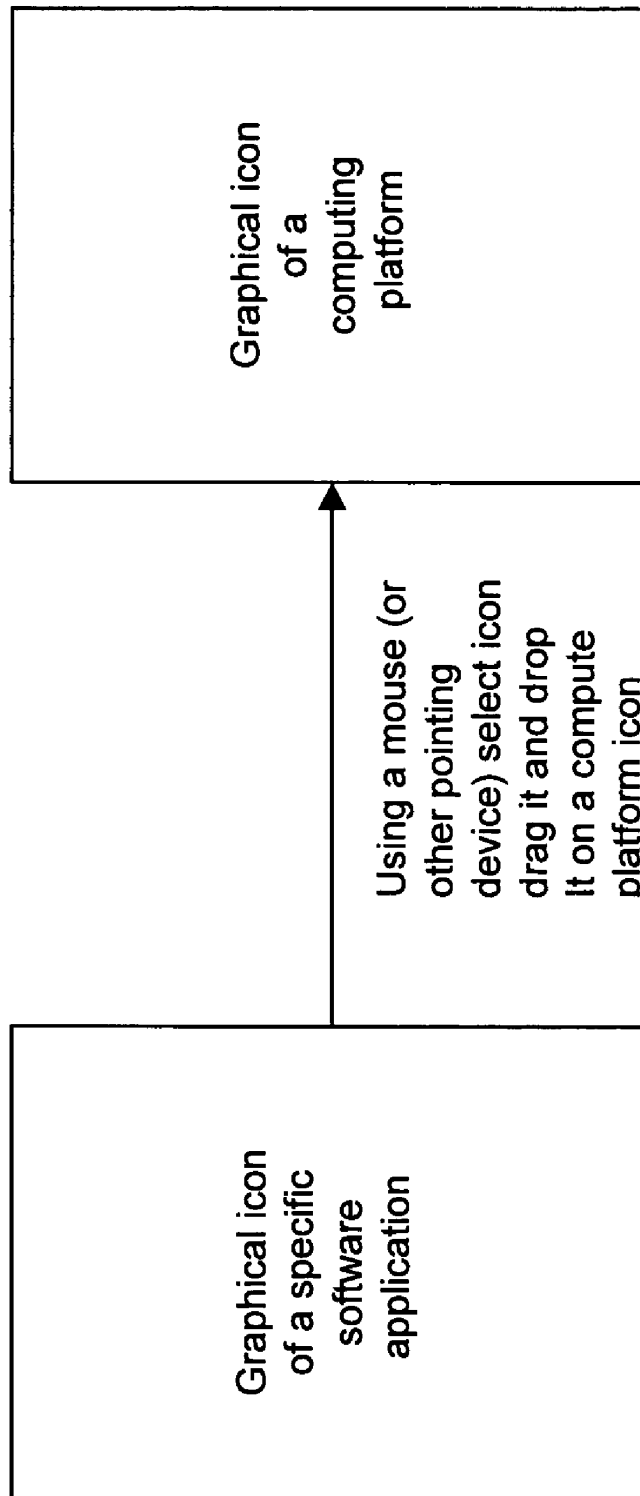


Figure 12

U.S. Patent

Apr. 14, 2009

Sheet 13 of 17

US 7,519,814 B2

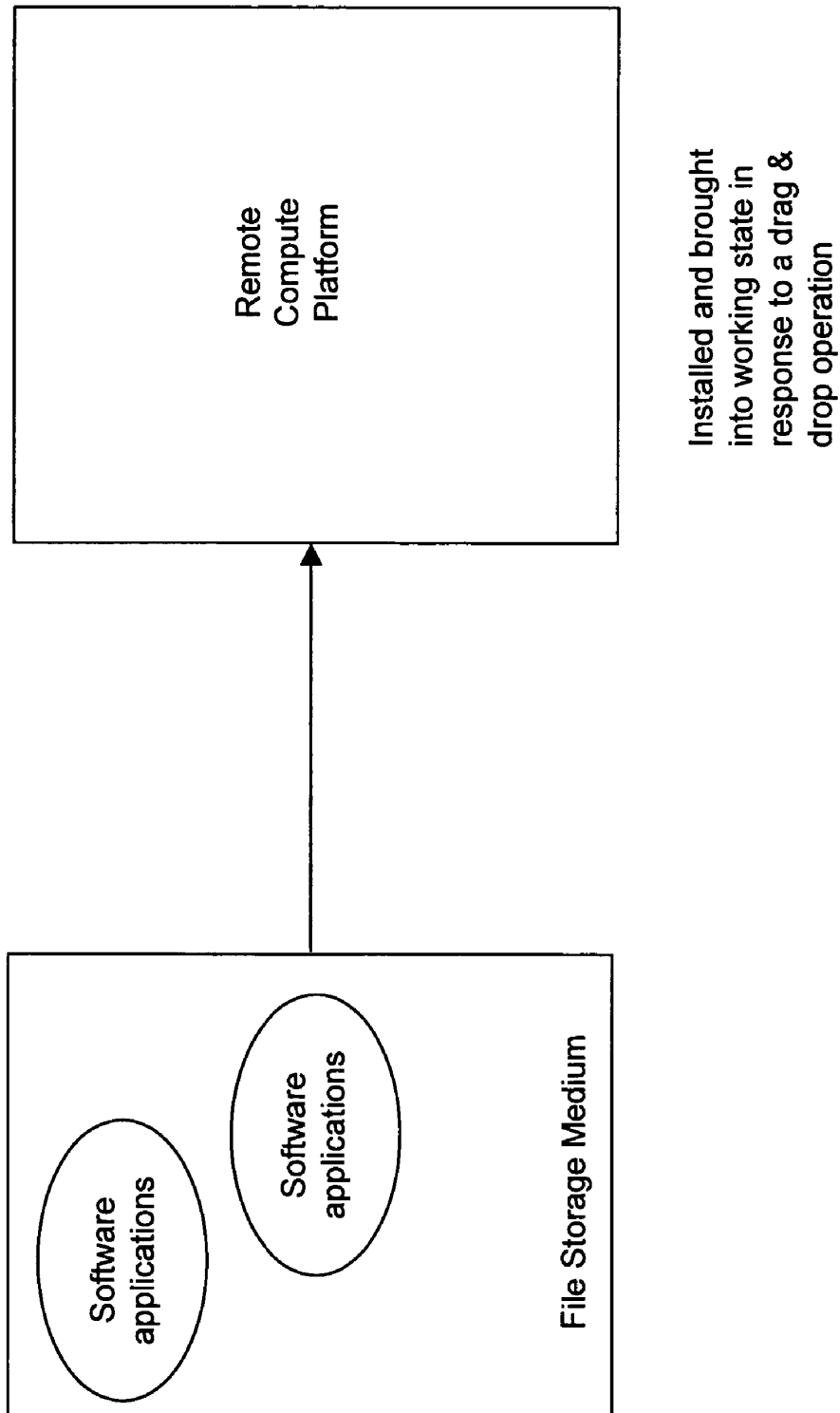


Figure 13

U.S. Patent

Apr. 14, 2009

Sheet 14 of 17

US 7,519,814 B2

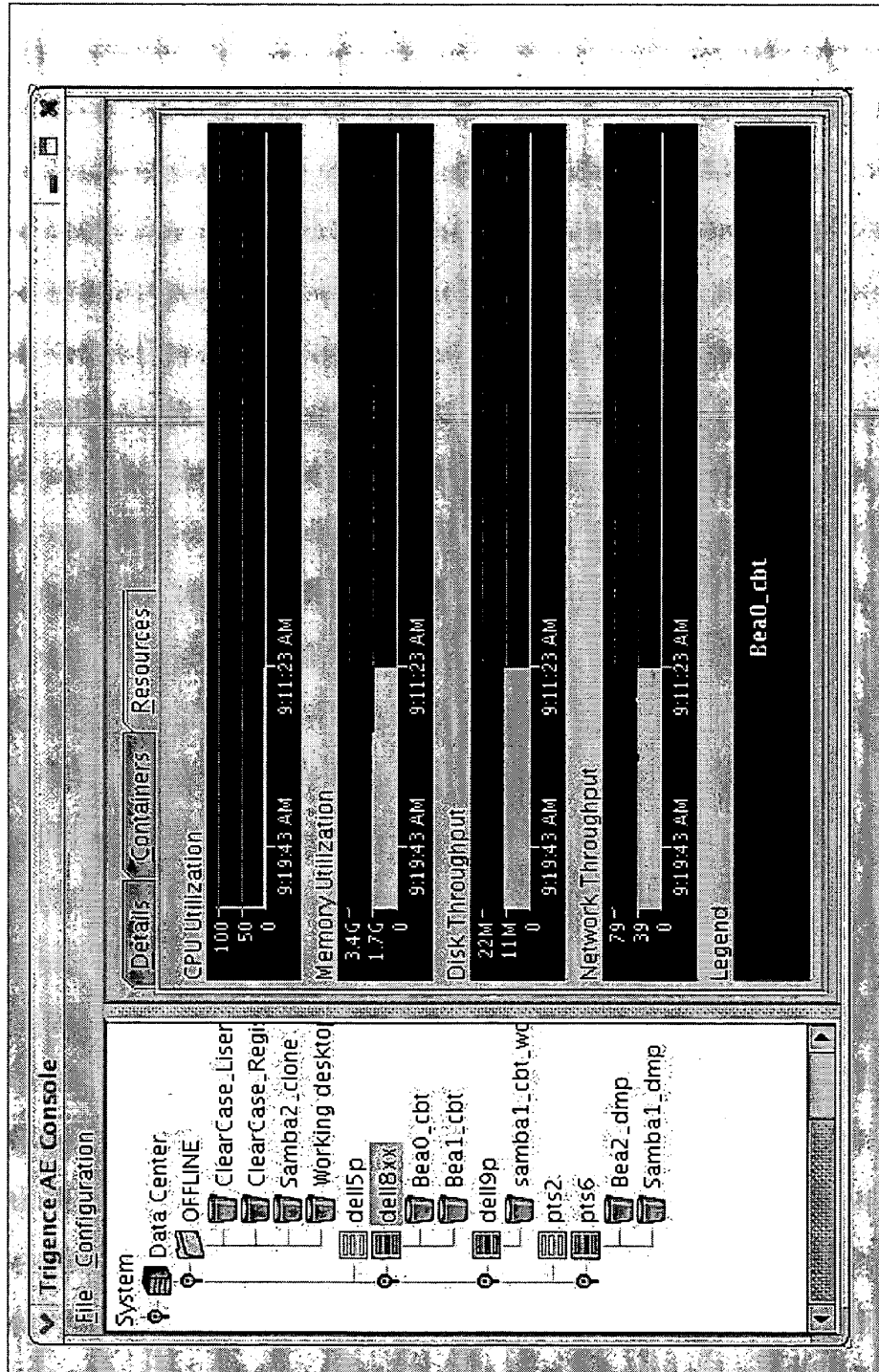


Figure 14

U.S. Patent

Apr. 14, 2009

Sheet 15 of 17

US 7,519,814 B2

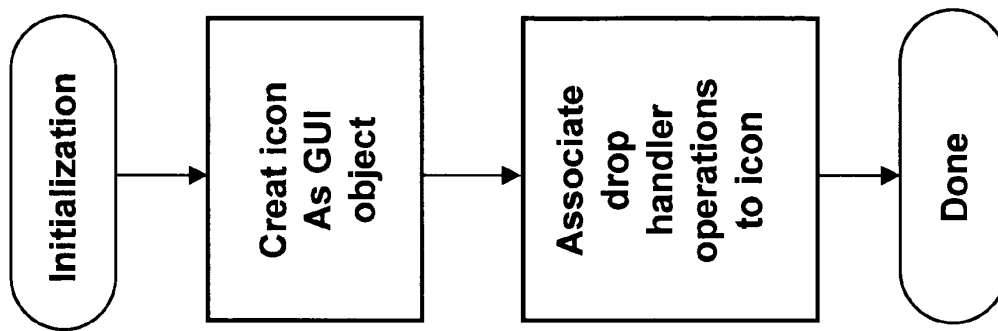


Figure 15

U.S. Patent

Apr. 14, 2009

Sheet 16 of 17

US 7,519,814 B2

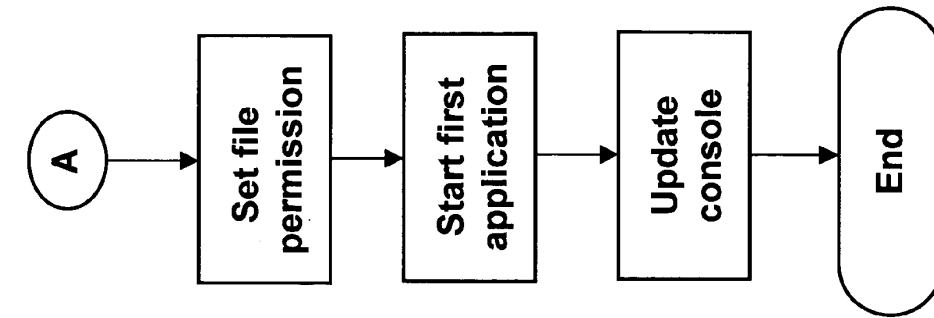
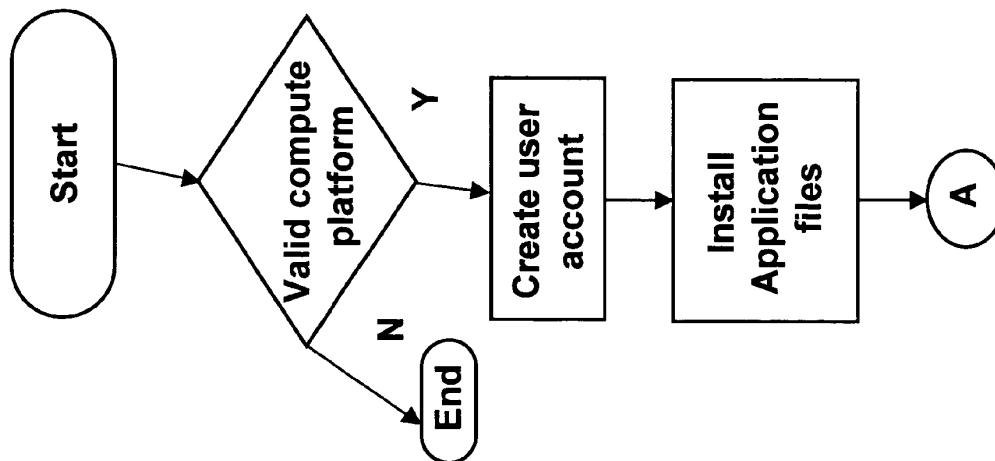


Figure 16



U.S. Patent

Apr. 14, 2009

Sheet 17 of 17

US 7,519,814 B2

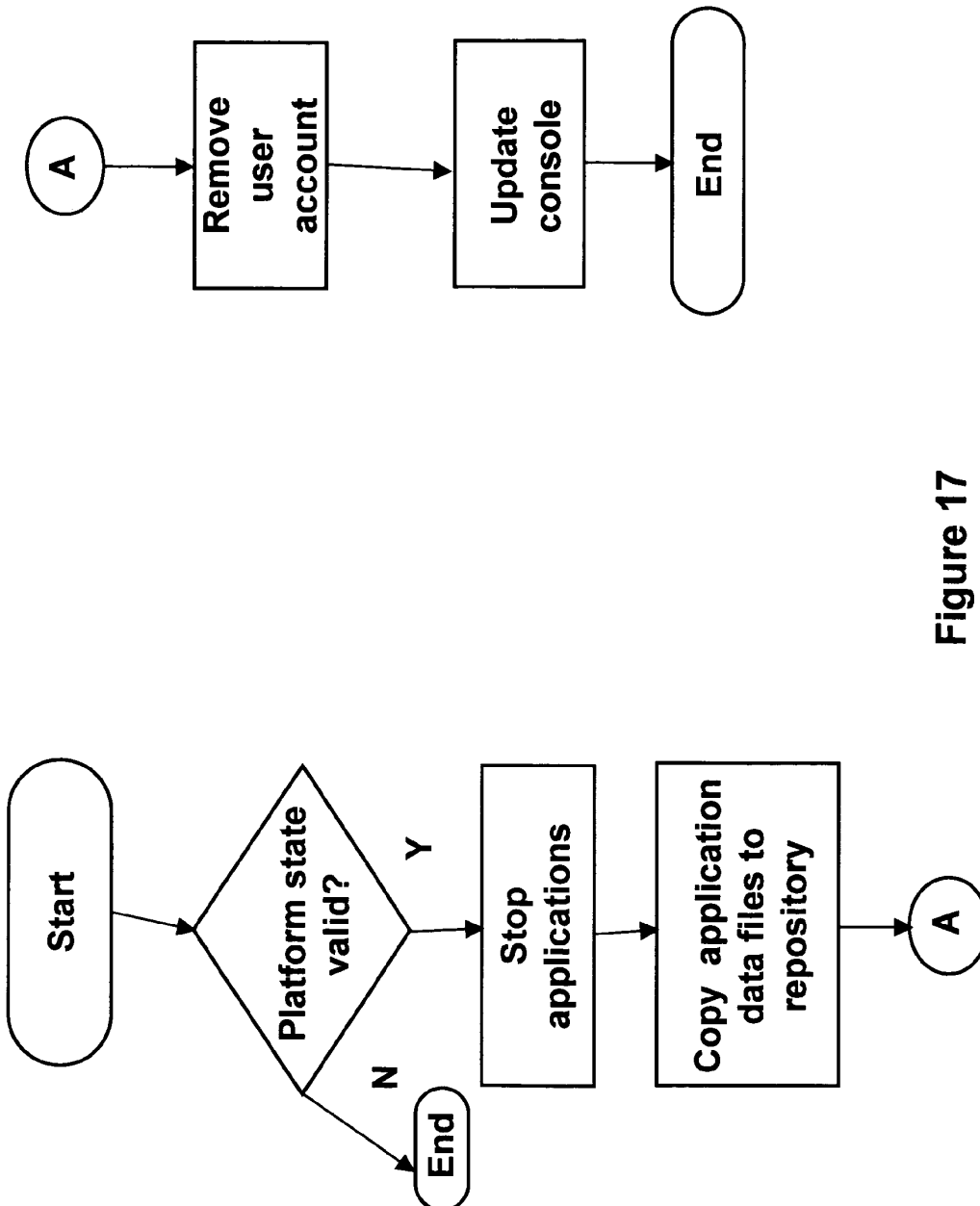


Figure 17

US 7,519,814 B2

1

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**CROSS-REFERENCE TO RELATED APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

2

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

```
/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1
```


US 7,519,814 B2

3

/lib/libnsl.so.1
 /lib/libdl.so.2
 /lib/i686/libpthread.so.0
 /lib/i686/libc.so.6
 /lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
 /etc/httpd/conf
 /etc/httpd/conf.d
 /etc/httpd/logs
 /etc/httpd/modules
 /etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

4

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySql and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

US 7,519,814 B2

5

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of

6

this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed "spoofing" and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. 3 is a pictorial diagram illustrating the creation of a container.

FIG. 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the "kernel module" which is used to handle system calls.

FIG. 5 is a diagram similar to FIG. 4, wherein additional configuration data is provided.

FIG. 6 is a diagram illustrating a system wherein the containers are secure containers.

FIG. 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. 9 is a diagram that illustrates the installation of a container on a server.

FIG. 10 is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. 14 is a screen snapshot of an implementation according to the schematic of FIG. 13.

FIG. 15 is a flow chart of a method of initializing icons of FIG. 14.

FIG. 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. 14; and,

FIG. 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13.

DETAILED DESCRIPTION

Turning now to FIG. 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c.

US 7,519,814 B2

7

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. 2 illustrates a system in accordance with the invention in which multiple applications **21a**, **21b**, **21c**, **21d**, **21e**, and **21f** can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20a**, **20b** and **20c**. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20a**, **20b** and **20c** are shown in FIG. 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. 4 illustrates a system having two secure containers **20a** **20b**, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20a**, **20b** environment system calls are intercepted, by the kernel module **43** passing through services installed as part of

8

this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

As introduced heretofore, “spoofing” is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system’s kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the ‘uname’ system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has ‘spoofed’ the ‘uname’ system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same “spoofing” mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

- 35 tracking applications, for example a tracking of which applications are in and out of a specific container;
- spoofing particular values returned by the kernel;
- applying resource checks and imposing resource limits;
- monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,
- 40 retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20a** or **20b**. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. 6 that each application executing associated with a container **20a** or **20b**, or contained within a

US 7,519,814 B2

9

container **20a** or **20b**, is able to access files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20a** association are not able to access the files contained in another container **20b**.

When a container **20a** or **20b** is installed specific applications are started, as is shown in FIG. 7 and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. 7, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. 8, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. 9 shows that the files **82a**, **84a** are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing

10

the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. 9.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform and then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. 3, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

US 7,519,814 B2

11

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

- modifying the contents a file to add a container specific IP address;
- modifying the contents of a directory to define start scripts that should be executed;
- modifying the contents of a file to define which mount points will relate to a container;
- removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications

12

are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

US 7,519,814 B2

13

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

14

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

US 7,519,814 B2

15

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. 14, a screen snapshot of an implementation is shown according to the schematic of FIG. 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. 15, shown is a flow chart of a method of initializing the icons of FIG. 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. 16 are performed by an install drop handler. During installation, verification is performed to

16

determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. 15), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

US 7,519,814 B2

17

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

18

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

- a) running an instance of a service on a server;
- b) determining which files are being used; and,
- c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim 14 comprising the steps of:

- assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
- installing the container on a server; and,
- testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

- using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with a unique icon representing

US 7,519,814 B2

19

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim 20 further comprising starting a distributed software application.

22. A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim 17 further comprising creating a user account for the service.

25. A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim 1, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;
displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

20

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim 31, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

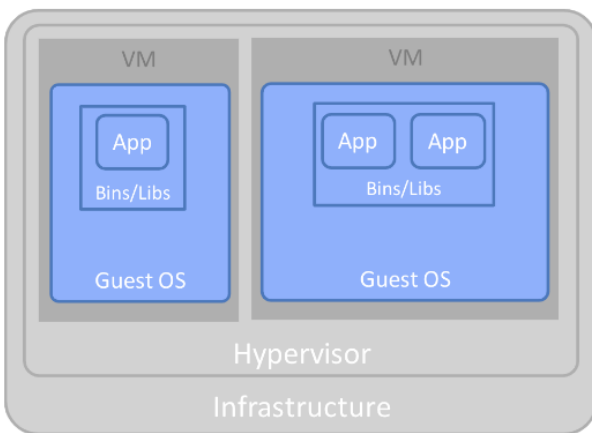
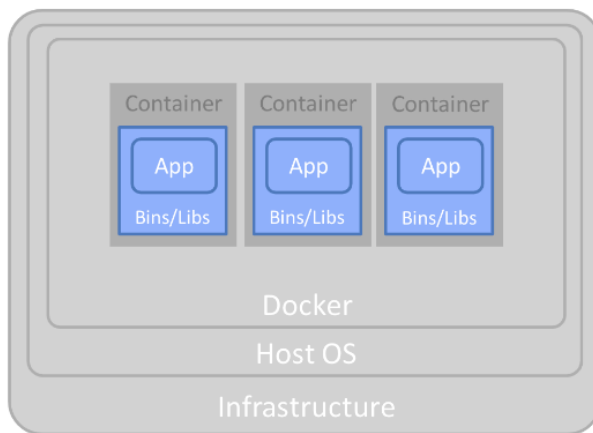
Exhibit 2

U.S. Patent No. 7,519,814 (“814 Patent”)

Accused Instrumentalities: IBM’s Cloud Kubernetes Service (IKS) and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>To the extent the preamble is limiting, IBM practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>IBM Cloud® Kubernetes Service provides a fully managed container service for Docker (OCI) containers, so clients can deploy containerized apps onto a pool of compute hosts and subsequently manage those containers. Containers are automatically scheduled and placed onto available compute hosts based on your requirements and availability in the cluster.</p> <p>https://www.ibm.com/products/kubernetes-service</p> <p>With IBM Cloud Kubernetes Service, you can deploy Docker containers into pods that run on your worker nodes. The worker nodes come with a set of add-on pods to help you manage your containers. Install more add-ons through Helm, a Kubernetes package manager. These add-ons can extend your apps with dashboards, logging, IBM Cloud and IBM Watson® services and more.</p> <p>https://www.ibm.com/products/kubernetes-service</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="661 332 1680 487">Docker is an open source platform that enables developers to build, deploy, run, update and manage <i>containers</i>—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.</p> <p data-bbox="661 511 1092 544">https://www.ibm.com/topics/docker</p> <div data-bbox="661 592 1858 1079"><div><p data-bbox="840 592 1081 625">Virtual Machines</p><p>The diagram shows two Virtual Machines (VMs) stacked vertically. Each VM contains a Guest OS, which in turn contains Bins/Libs and an App. The VMs are managed by a Hypervisor, which sits on top of the Infrastructure layer.</p></div><div><p data-bbox="1491 592 1648 625">Containers</p><p>The diagram shows three Containers stacked vertically. Each Container contains Bins/Libs and an App. The Containers are managed by Docker, which sits on top of the Host OS, which in turn sits on top of the Infrastructure layer.</p></div></div> <p data-bbox="661 1112 1564 1144">https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/</p>

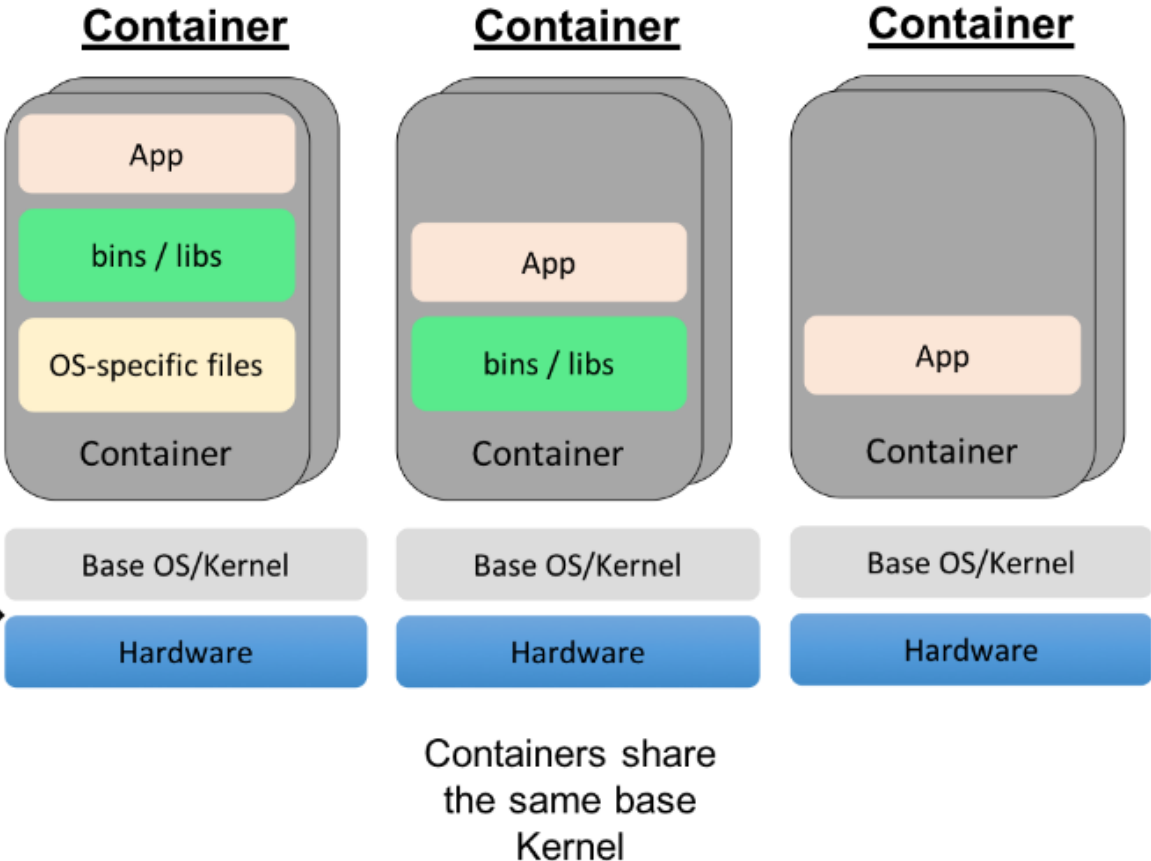
Claim 1	Accused Instrumentalities
	<p>Containers are executable units of software in which application code is packaged along with its libraries and dependencies, in common ways so that the code can be run anywhere—whether it be on desktop, traditional IT or the cloud.</p> <p>To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (e.g. Linux namespaces and cgroups, Windows silos and job objects) can be leveraged to isolate processes and control the amount of CPU, memory and disk that those processes can access.</p> <p>Containers are small, fast and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can instead simply leverage the features and resources of the host OS.</p> <p>https://www.ibm.com/topics/containers</p> <p>With containers, you can isolate the ecosystem to run an application on any host OS (operating system). Containers can wrap code, runtimes, system tools, system libraries—everything that can be installed on a server. Containers are like virtual machines (VMs), but with a key difference in their architectural approach. Images that run on VMs have a full copy of the guest OS, including the necessary binaries and libraries. Images that run on containers share the OS kernel on the host.</p> <p>The Docker Engine builds and spins images on the containers. The engine is a lightweight container runtime that can run on almost any OS. You can run a container anywhere that a Docker Engine can be installed—on bare metal servers, clouds, and even inside a VM. You can move containers from one environment to another without recoding the application.</p> <p>Containers can help DevOps teams in three ways:</p> <ul style="list-style-type: none"> • Increase development productivity by reducing the time spent on environment setup • Eliminate issues that are caused by software dependencies • Avoid inconsistencies when applications are run in different environments <p>You can use IBM Cloud Kubernetes Service to run containers on IBM Cloud.</p> <p>https://www.ibm.com/garage/method/practices/run/tool_ibm_container/, last accessed on Nov. 17, 2023.</p>

Claim 1	Accused Instrumentalities
	<p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p> <p>Today Docker containerization also works with Microsoft Windows and Apple MacOS. Developers can run Docker containers on any operating system, and most leading cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to help developers build, deploy and run applications containerized with Docker.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="657 329 1799 1187"> <p>The diagram illustrates three containers, each labeled 'Container' at the bottom. Each container is a stack of three colored boxes: an orange box labeled 'App' at the top, a green box labeled 'bins / libs' in the middle, and a yellow box labeled 'OS-specific files' at the bottom. These containers are stacked on a grey box labeled 'Base OS/Kernel', which is itself on a blue box labeled 'Hardware'. Below the containers, the text 'Containers share the same base Kernel' is displayed, followed by the URL https://ibm.github.io/kube101/.</p> </div>
[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application	The method practiced by IBM through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of

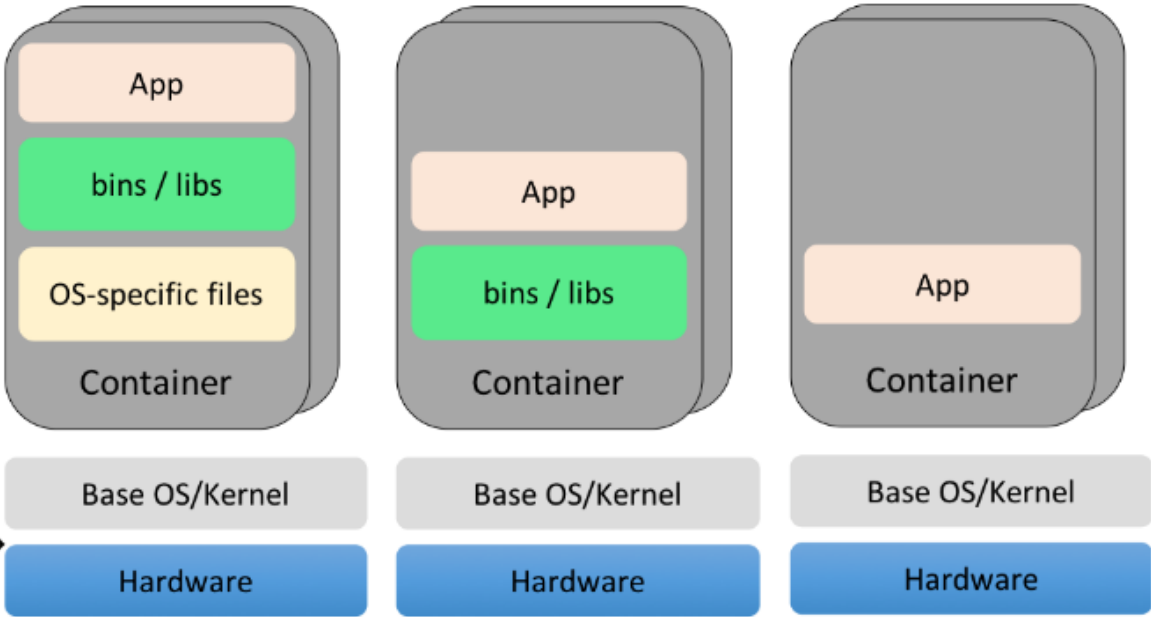
Claim 1	Accused Instrumentalities
<p>software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p> <p><i>See, e.g.:</i></p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p> <p>Today Docker containerization also works with Microsoft Windows and Apple MacOS. Developers can run Docker containers on any operating system, and most leading cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to help developers build, deploy and run applications containerized with Docker.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<div style="text-align: center;">  <p>Containers share the same base Kernel</p> <p>https://ibm.github.io/kube101/</p> </div>
[1b] wherein the set of associated system files are compatible with a local kernel	In the method practiced by IBM through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.

Claim 1	Accused Instrumentalities
of at least some of the plurality of different operating systems,	<p data-bbox="653 321 766 354"><i>See, e.g.:</i></p> <p data-bbox="653 386 1833 638">Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p data-bbox="653 670 1839 971">Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.</p> <p data-bbox="653 987 1201 1019">https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="657 331 1801 1187"> <p>The diagram illustrates three containers, each labeled 'Container' at the bottom. Each container is a stack of three colored boxes: an orange box labeled 'App' at the top, a green box labeled 'bins / libs' in the middle, and a yellow box labeled 'OS-specific files' at the bottom. These containers are stacked on a single gray box labeled 'Base OS/Kernel'. Below this is a blue box labeled 'Hardware'. The text 'Containers share the same base Kernel' is centered below the containers.</p> <p>https://ibm.github.io/kube101/</p> </div>
[1c] the containers of application software excluding a kernel,	<p>In the method practiced by IBM through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<div style="text-align: center;"> <p><u>Container</u> <u>Container</u> <u>Container</u></p>  <p>Containers share the same base Kernel</p> <p>https://ibm.github.io/kube101/</p> </div>
[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the	In the method practiced by IBM through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.

Claim 1	Accused Instrumentalities
associated local system files that remain resident on the server,	<p><i>See, e.g.:</i></p> <p>Rather than spinning up an entire virtual machine, containerization packages together everything needed to run a single application or microservice (along with runtime libraries they need to run). The container includes all the code, its dependencies and even the operating system itself. This enables applications to run almost anywhere — a desktop computer, a traditional IT infrastructure or the cloud.</p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p>
[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,	<p>In the method practiced by IBM through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.</p> <p>Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<p>With containers, you can isolate the ecosystem to run an application on any host OS (operating system). Containers can wrap code, runtimes, system tools, system libraries—everything that can be installed on a server. Containers are like virtual machines (VMs), but with a key difference in their architectural approach. Images that run on VMs have a full copy of the guest OS, including the necessary binaries and libraries. Images that run on containers share the OS kernel on the host.</p> <p>The Docker Engine builds and spins images on the containers. The engine is a lightweight container runtime that can run on almost any OS. You can run a container anywhere that a Docker Engine can be installed—on bare metal servers, clouds, and even inside a VM. You can move containers from one environment to another without recoding the application.</p> <p>Containers can help DevOps teams in three ways:</p> <ul style="list-style-type: none"> • Increase development productivity by reducing the time spent on environment setup • Eliminate issues that are caused by software dependencies • Avoid inconsistencies when applications are run in different environments <p>You can use IBM Cloud Kubernetes Service to run containers on IBM Cloud.</p> <p>https://www.ibm.com/garage/method/practices/run/tool_ibm_container/, last accessed on Nov. 17, 2023.</p>
[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,	<p>In the method practiced by IBM through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p><i>See, e.g.:</i></p> <p>Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple virtual machines (VMs) to share the CPU, memory and other resources of a single hardware server.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p>Fault isolation: Each containerized application is isolated and operates independently of others. The failure of one container does not affect the continued operation of any other containers. Development teams can identify and correct any technical issues within one container without any downtime in other containers. Also, the container engine can leverage any OS security isolation techniques—such as SELinux access control—to isolate faults within containers.</p> <p>https://www.ibm.com/topics/containerization</p>
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p><i>See, e.g.:</i></p> <p>Limit the number of privileged containers. Containers run as a separate Linux process on the compute host that is isolated from other processes. Although users have root access inside the container, the permissions of this user are limited outside the container to protect other Linux processes, the host file system, and host devices. Some apps require access to the host file system or advanced permissions to run properly. You can run containers in privileged mode to allow the container the same access as the processes running on the compute host. Keep in mind that privileged containers can cause huge damage to the cluster and the underlying compute host if they become compromised. Try to limit the number of containers that run in privileged mode and consider changing the configuration for your app so that the app can run without advanced permissions.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-security</p>

Claim 1	Accused Instrumentalities
	<p>Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple <i>virtual machines</i> (VMs) to share the CPU, memory and other resources of a single hardware server.</p> <p>https://www.ibm.com/topics/docker</p>

Exhibit 3



(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,784,058 B2**
(45) **Date of Patent:** **Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**

2002/0004854 A1 1/2002 Hartley
2002/0174215 A1 11/2002 Schaefer 709/224
2003/0101292 A1 5/2003 Fisher
2004/0025165 A1* 2/2004 Desoli et al. 719/310

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002
WO WO 2006/039181 A 4/2006

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

OTHER PUBLICATIONS

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1293 days.

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

* cited by examiner

(21) Appl. No.: **10/946,536**

Primary Examiner—Hyung S Sough

Assistant Examiner—Syed Roni

(22) Filed: **Sep. 21, 2004**

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A. Attorneys at Law

(65) **Prior Publication Data**

US 2005/0066303 A1 Mar. 24, 2005

(57) **ABSTRACT**

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep. 22, 2003.

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

(51) **Int. Cl.**
G06F 9/22 (2006.01)

(52) **U.S. Cl.** **719/310**; 719/319

(58) **Field of Classification Search** 719/310,
719/319

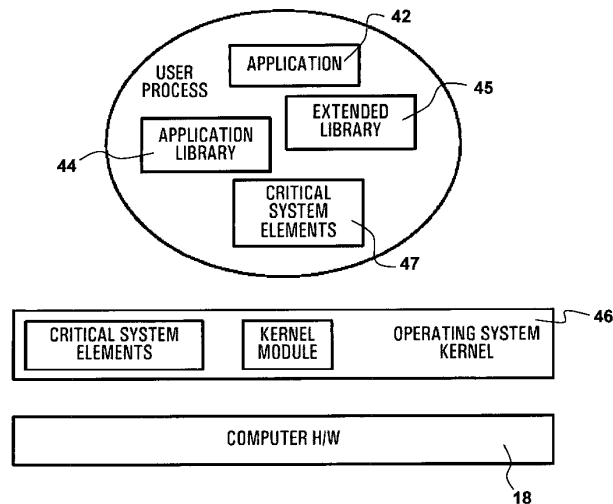
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,481,706 A * 1/1996 Peek 710/200
6,212,574 B1 * 4/2001 O'Rourke et al. 719/321
6,260,075 B1 * 7/2001 Cabrero et al. 719/310

18 Claims, 7 Drawing Sheets



U.S. Patent

Aug. 24, 2010

Sheet 1 of 7

US 7,784,058 B2

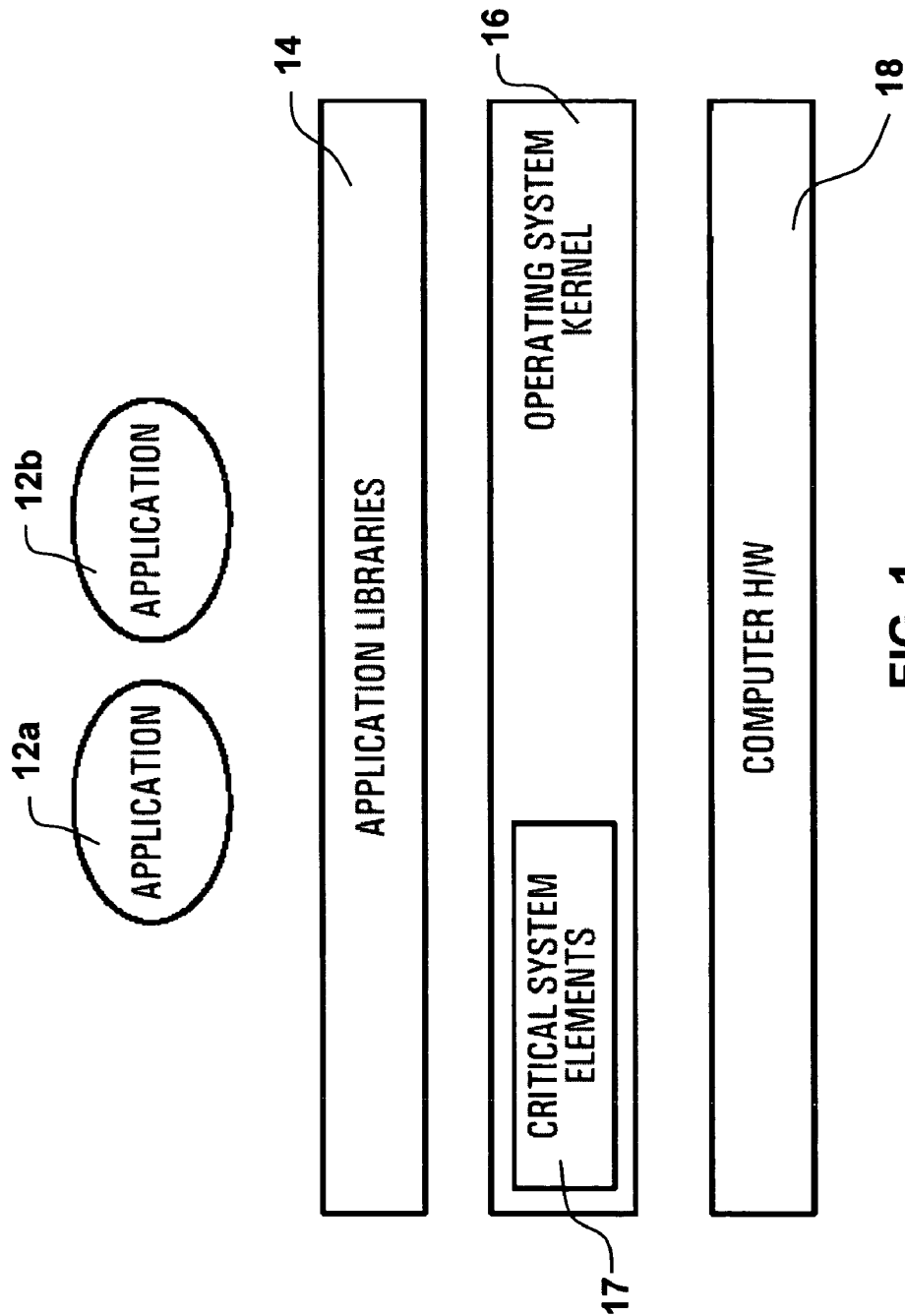


FIG. 1
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 2 of 7

US 7,784,058 B2

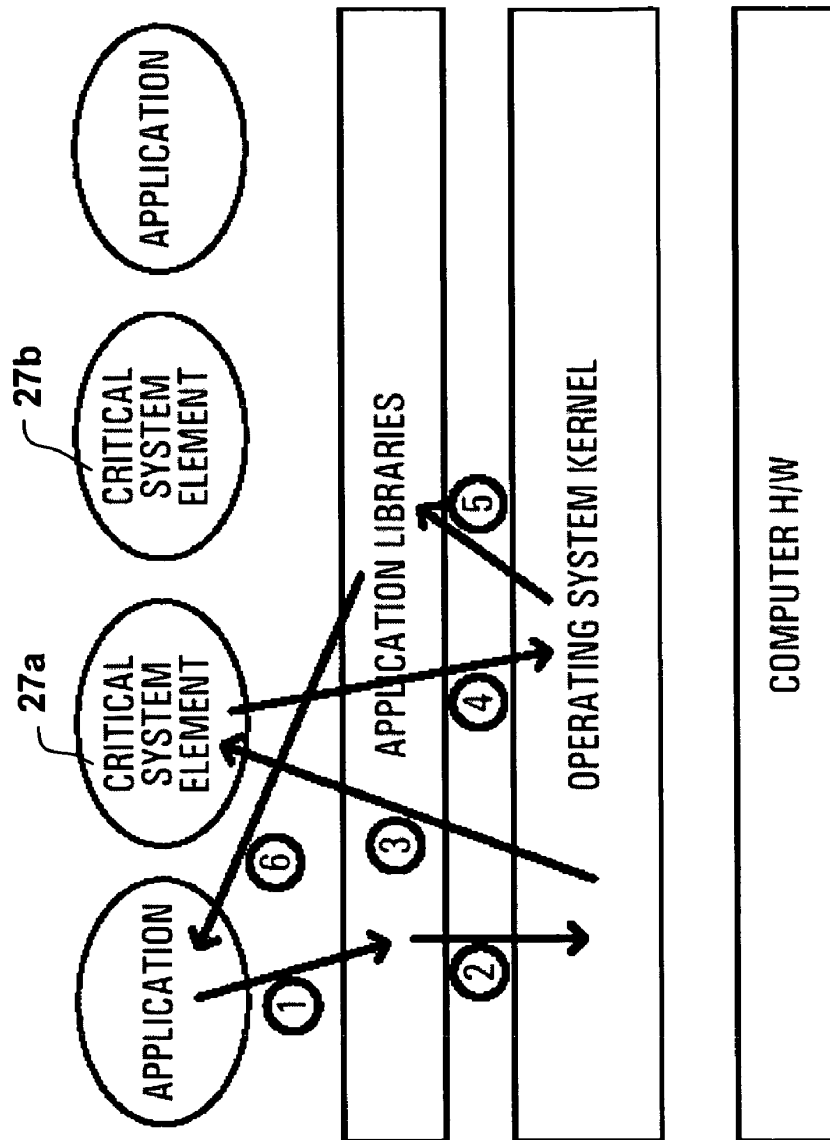


FIG. 2a
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 3 of 7

US 7,784,058 B2

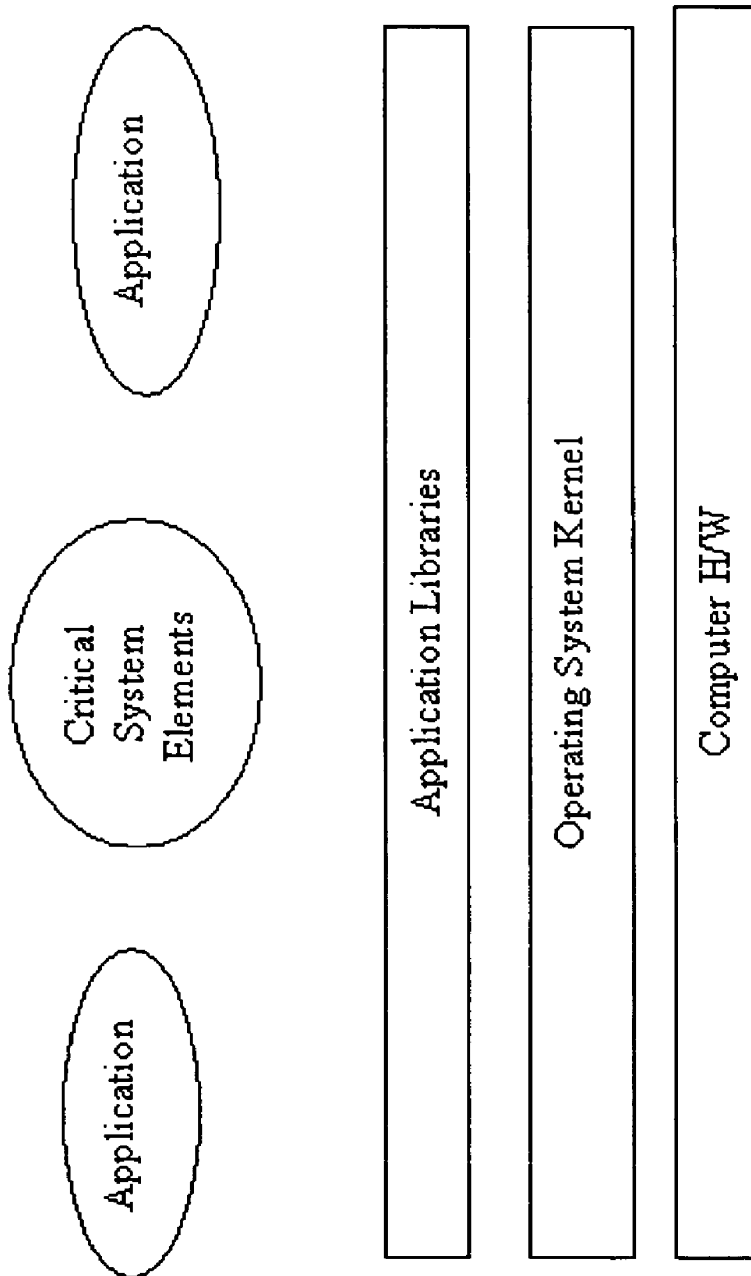


FIG. 2b
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 4 of 7

US 7,784,058 B2

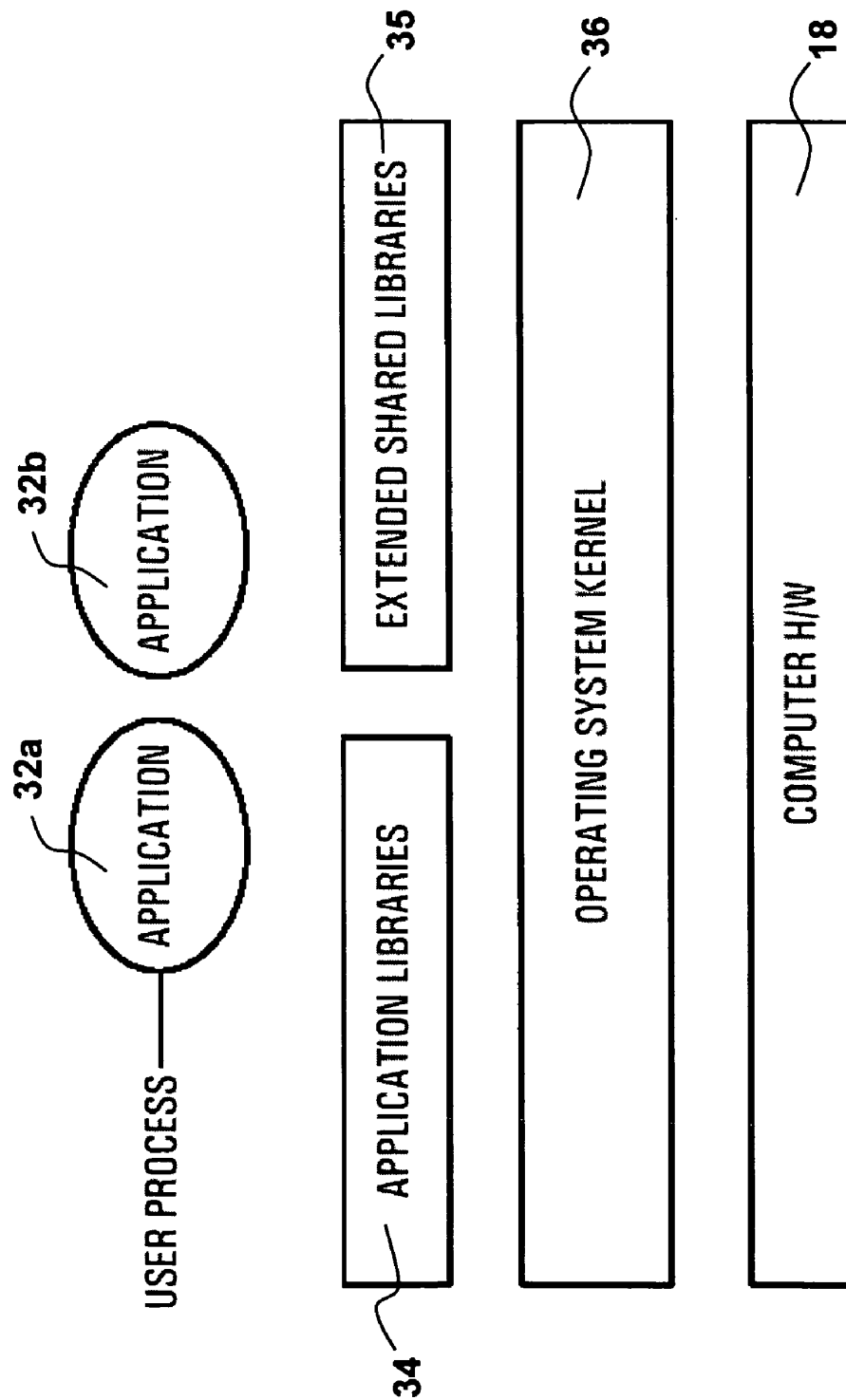


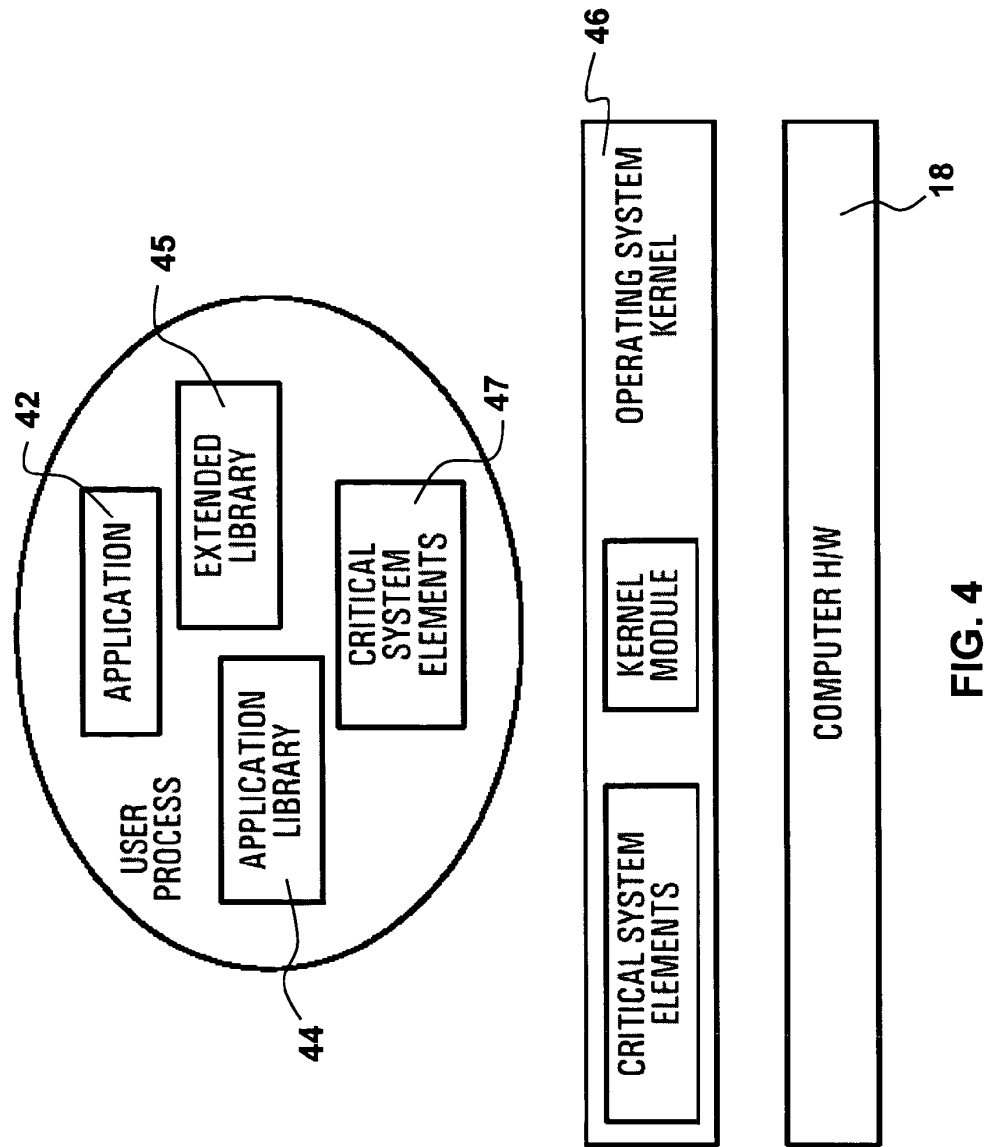
FIG. 3

U.S. Patent

Aug. 24, 2010

Sheet 5 of 7

US 7,784,058 B2



U.S. Patent

Aug. 24, 2010

Sheet 6 of 7

US 7,784,058 B2

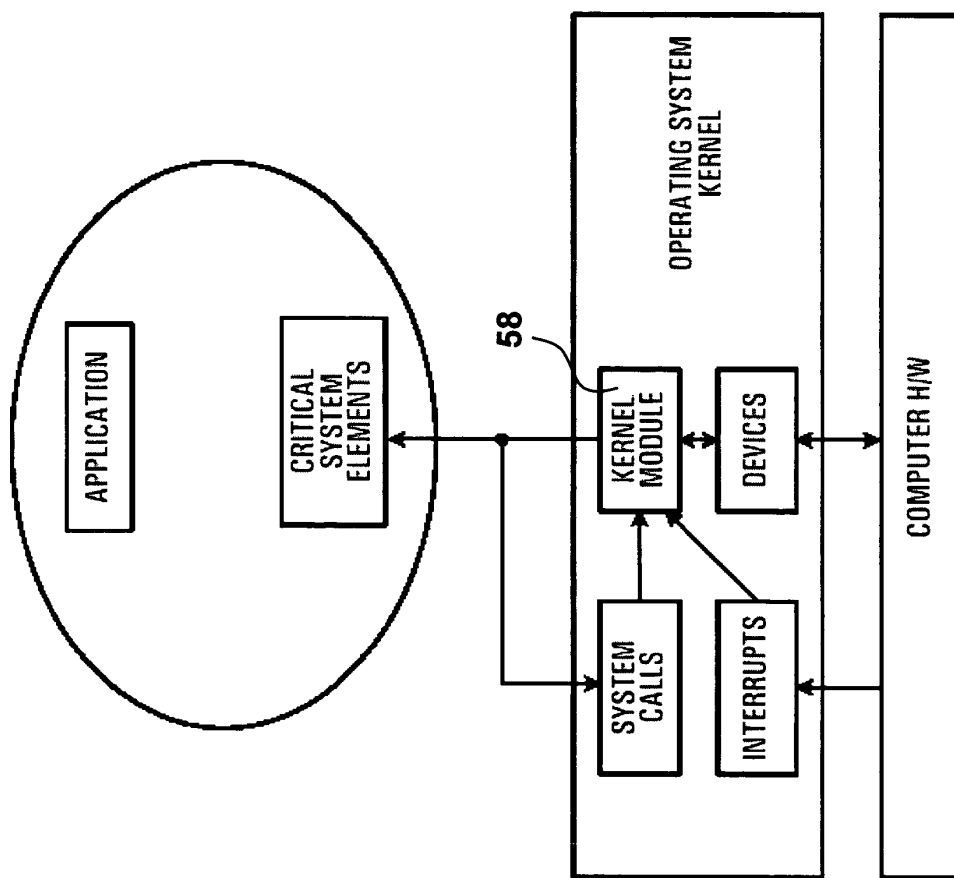


FIG. 5

U.S. Patent

Aug. 24, 2010

Sheet 7 of 7

US 7,784,058 B2

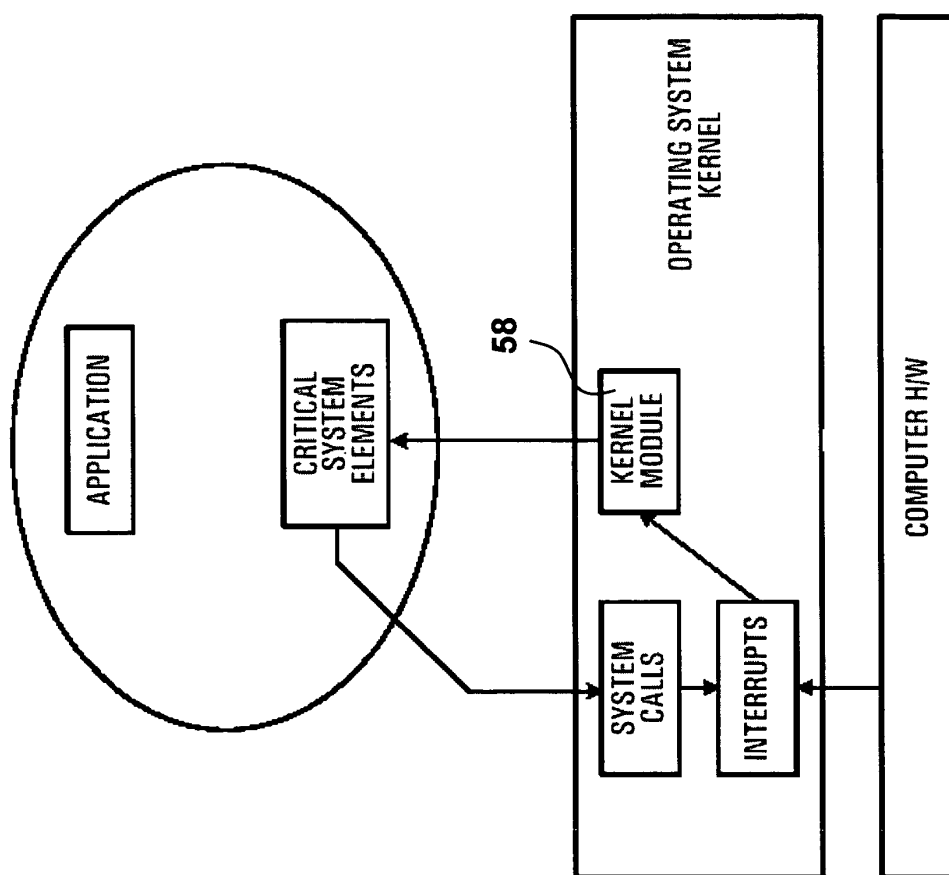


FIG. 6

US 7,784,058 B2

1

**COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled “User Mode Critical System Element as Shared Libs”, which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

2

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.

2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

US 7,784,058 B2

3

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

4

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. 1 is an architectural view of the traditional monolithic prior art operating system;

US 7,784,058 B2

5

FIG. 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. 3 is an architectural view of an embodiment of the invention;

FIG. 4 is a functional view showing how critical system elements exist in the same context as an application;

FIG. 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. 6 shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TCP/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP

6

stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality

2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided

2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b

US 7,784,058 B2

7

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12a**, **12b** makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12a**, **12b** in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2a** shows a system architecture where critical system elements **27a**, **27b** execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2a** is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2a** as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2b** is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2a** and FIG. **2b** is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2b** is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the

8

case with a multi-server architecture depicted in FIG. **2a**. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32a**, **32b**, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45,47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

US 7,784,058 B2

9

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. 4. FIG. 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

10

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:
 - a) a processor;
 - b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,
 - c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

US 7,784,058 B2

11

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and
iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.
2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.
3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.
4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.
5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.
6. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.
7. A computing system according to claim 6 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.
8. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

12

9. A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.
10. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.
11. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.
12. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.
13. A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.
14. A computing system according to claim 13 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.
15. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.
16. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.
17. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.
18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.

* * * * *

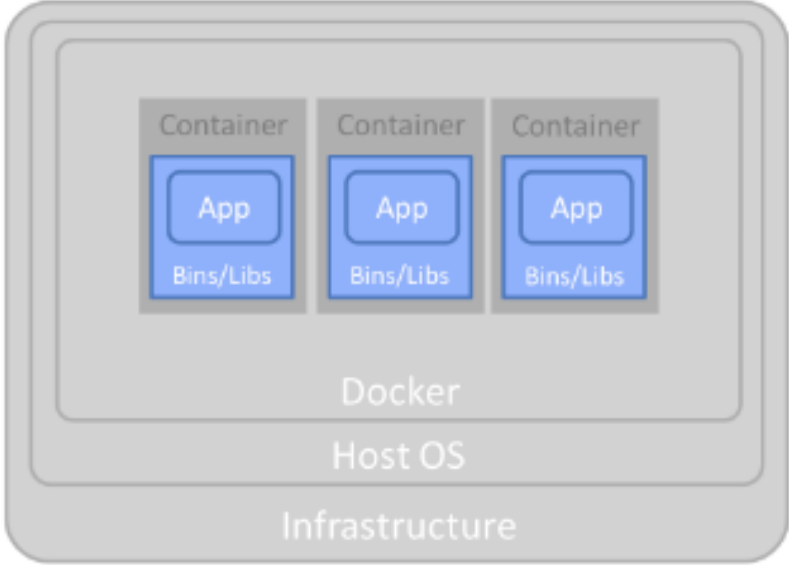
Exhibit 4

U.S. Patent No. 7,784,058 (“’058 Patent”)

Accused Instrumentalities: IBM’s IBM Cloud Kubernetes Service, and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
<p>[1pre] 1. A computing system for executing a plurality of software applications comprising:</p>	<p>To the extent the preamble is limiting, each Accused Instrumentality comprises or constitutes a computing system for executing a plurality of software applications as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>IBM Cloud® Kubernetes Service provides a fully managed container service for Docker (OCI) containers, so clients can deploy containerized apps onto a pool of compute hosts and subsequently manage those containers. Containers are automatically scheduled and placed onto available compute hosts based on your requirements and availability in the cluster.</p> <p>https://www.ibm.com/products/kubernetes-service</p> <p>With IBM Cloud Kubernetes Service, you can deploy Docker containers into pods that run on your worker nodes. The worker nodes come with a set of add-on pods to help you manage your containers. Install more add-ons through Helm, a Kubernetes package manager. These add-ons can extend your apps with dashboards, logging, IBM Cloud and IBM Watson® services and more.</p> <p>https://www.ibm.com/products/kubernetes-service</p>

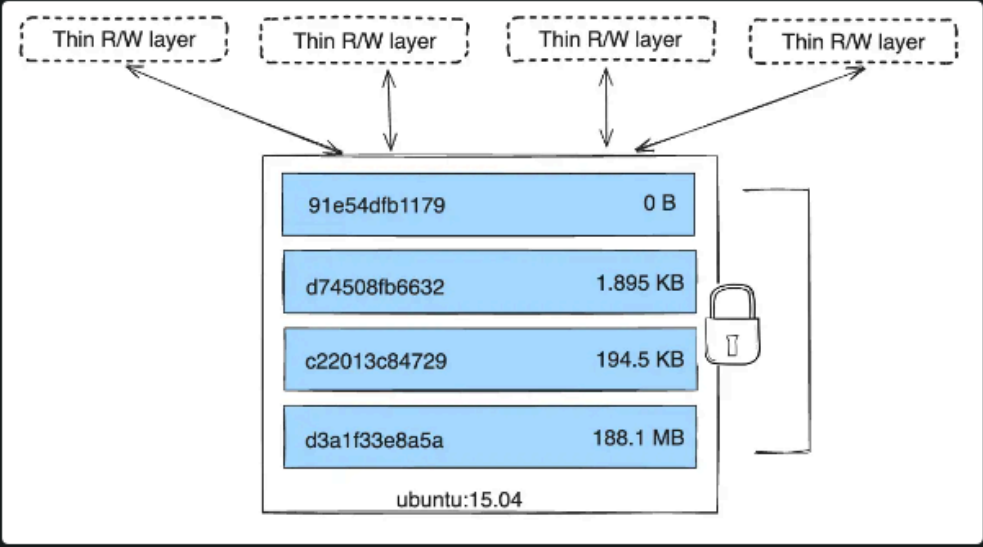
Claim 1	Accused Instrumentalities
	<p data-bbox="961 358 1163 402">Containers</p>  <p>The diagram illustrates a container architecture stack. At the base is a grey rounded rectangle labeled 'Infrastructure'. Above it is a grey rounded rectangle labeled 'Host OS'. Above the Host OS is a grey rounded rectangle labeled 'Docker'. Inside the Docker container are three separate grey rounded rectangles, each labeled 'Container'. Each 'Container' contains a blue rounded rectangle labeled 'App' and a smaller blue rounded rectangle labeled 'Bins/Libs' below it.</p> <p data-bbox="655 1073 1568 1105">https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/</p>
[1a] a) a processor;	<p data-bbox="655 1141 1299 1174">Each Accused Instrumentality comprises a processor.</p> <p data-bbox="655 1203 764 1235"><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containers are executable units of software in which application code is packaged along with its libraries and dependencies, in common ways so that the code can be run anywhere—whether it be on desktop, traditional IT or the cloud.</p> <p>To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (e.g. Linux namespaces and cgroups, Windows silos and job objects) can be leveraged to isolate processes and control the amount of CPU, memory and disk that those processes can access.</p> <p>Containers are small, fast and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can instead simply leverage the features and resources of the host OS.</p> <p>https://www.ibm.com/topics/containers</p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p>

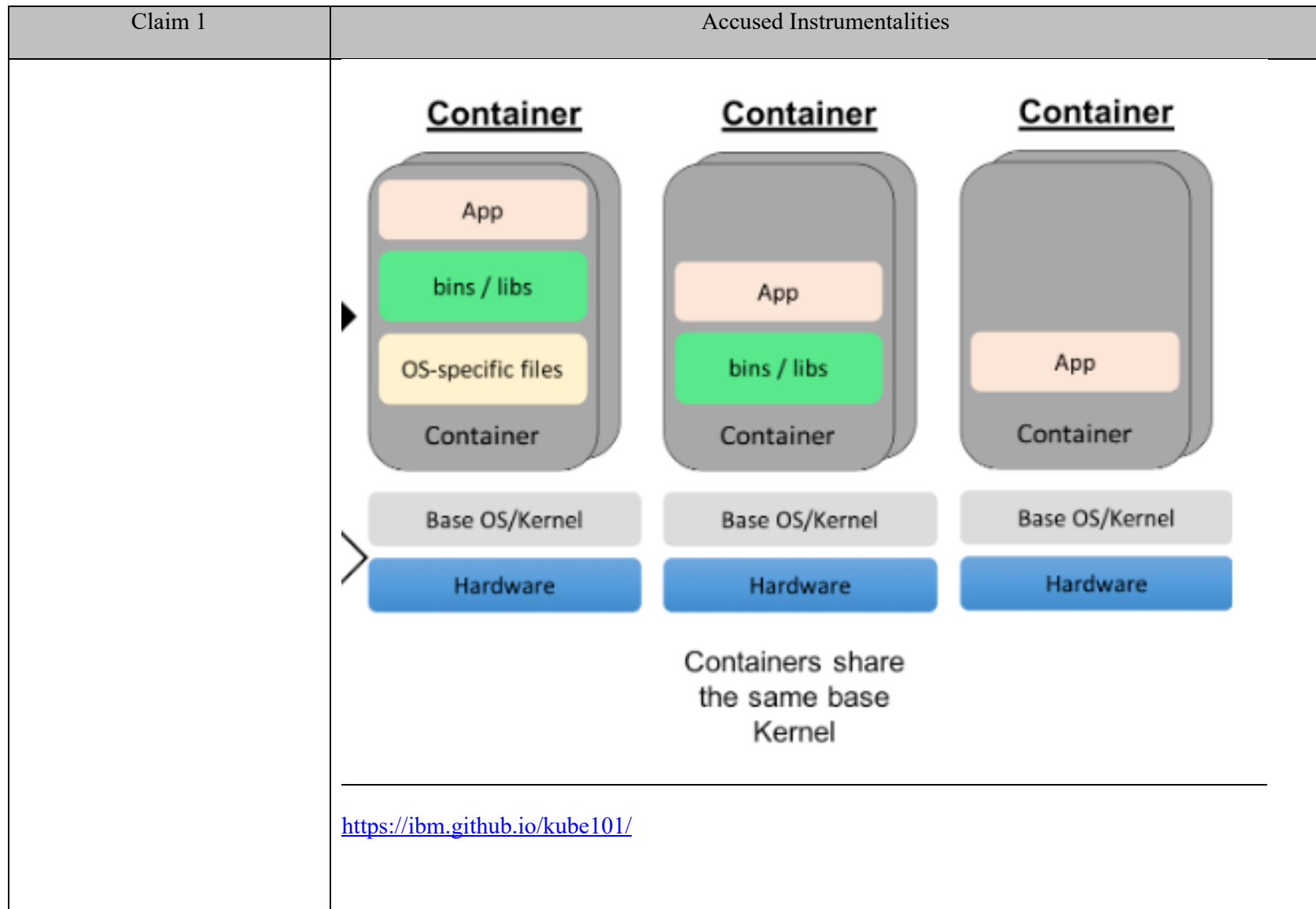
Claim 1	Accused Instrumentalities
<p>[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,</p>	<p>Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.</p> <p><i>See, e.g.:</i></p> <p>Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.</p> <p>Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.</p> <p>https://www.ibm.com/topics/containerization</p> <p>Kernel mode</p> <p>Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p>https://www.techtarget.com/searchdatacenter/definition/kernel</p>

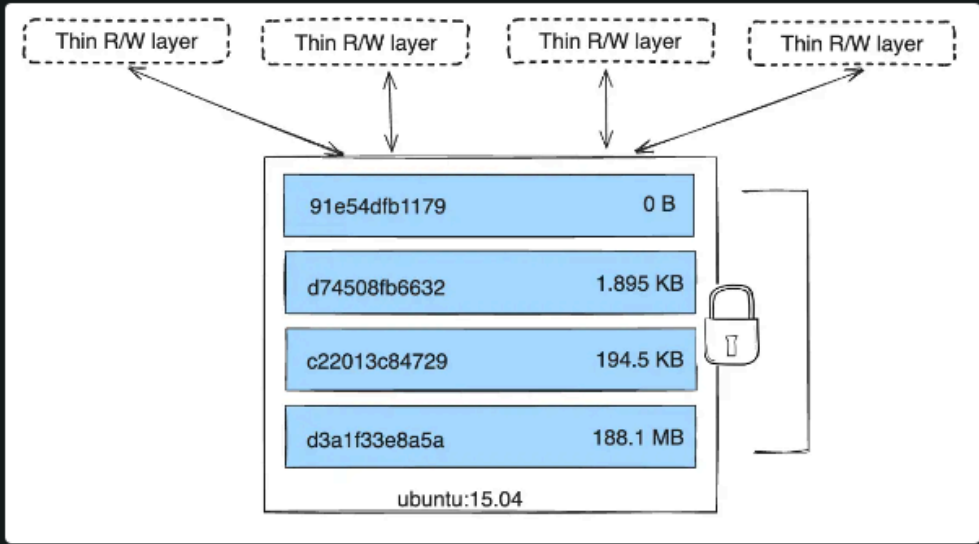
Claim 1	Accused Instrumentalities
<p>[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and</p>	<p>Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.</p> <p><i>See, e.g.:</i></p> <p>A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p>An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-images</p>

Claim 1	Accused Instrumentalities		
	Docker base image	Supported versions	Source of security notices
	Alpine	All stable versions with vendor security support.	Alpine SecDB database  .
	Debian	All stable versions with vendor security support. CVEs on binary packages that are associated with the Debian source package <code>linux</code> , such as <code>linux-libc-dev</code> , are not reported. Most of these binary packages are kernel and kernel modules, which are not run in container images.	Debian Security Bug Tracker  .
	GoogleContainerTools distroless	All stable versions with vendor security support.	GoogleContainerTools distroless  .
	Red Hat® Enterprise Linux® (RHEL)	RHEL/UBI 7, RHEL/UBI 8, and RHEL/UBI 9	Red Hat Security Data API  .
	Ubuntu	All stable versions with vendor security support.	Ubuntu CVE Tracker  .
	https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&interface=ui		

Claim 1	Accused Instrumentalities
	<p data-bbox="667 345 1864 467">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p data-bbox="653 1127 1213 1159">https://docs.docker.com/storage/storagedriver/</p>





Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>Docker images are made up of layers, and each layer corresponds to a version of the image. Whenever a developer makes changes to the image, a new top layer is created, and this top layer replaces the previous top layer as the current version of the image. Previous layers are saved for rollbacks or to be re-used in other projects.</p> <p>Each time a container is created from a Docker image, yet another new layer called the container layer is created. Changes made to the container—such as the addition or deletion of files—are saved to the container layer only and exist only while the container is running. This iterative image-creation process enables increased overall efficiency since multiple live container instances can run from just a single base image, and when they do so, they leverage a common stack.</p> <p>https://www.ibm.com/topics/docker</p>



Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>
[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of	In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.

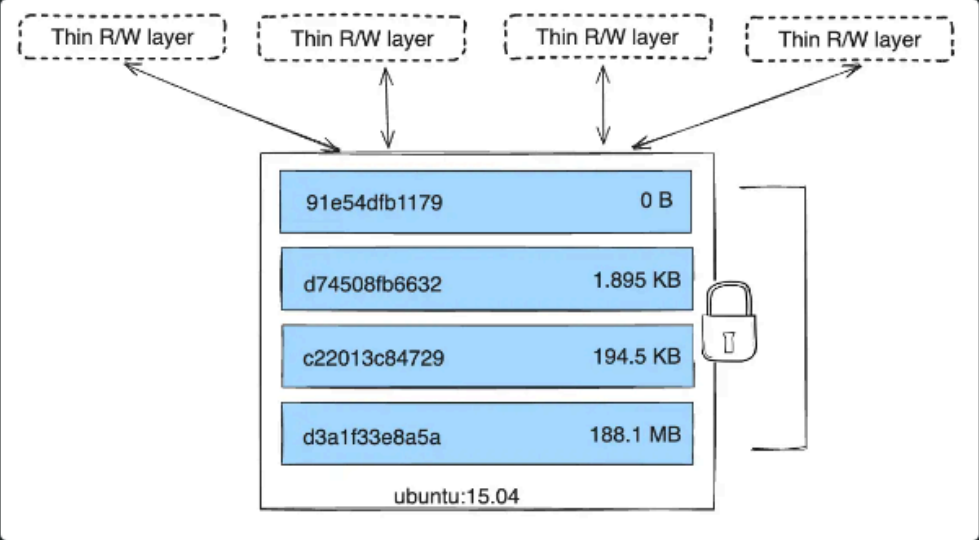
Claim 1	Accused Instrumentalities
<p>software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,</p>	<p>For example, a Docker base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and dependencies, including system libraries or shared library critical system elements (SLCSEs). When the container runs the image, it creates a runtime instance of that container image.</p> <p><i>See, e.g.:</i></p> <p>A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p>An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-images</p>

Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>Docker images are made up of layers, and each layer corresponds to a version of the image. Whenever a developer makes changes to the image, a new top layer is created, and this top layer replaces the previous top layer as the current version of the image. Previous layers are saved for rollbacks or to be re-used in other projects.</p> <p>Each time a container is created from a Docker image, yet another new layer called the container layer is created. Changes made to the container—such as the addition or deletion of files—are saved to the container layer only and exist only while the container is running. This iterative image-creation process enables increased overall efficiency since multiple live container instances can run from just a single base image, and when they do so, they leverage a common stack.</p> <p>https://www.ibm.com/topics/docker</p> <p>Following software is installed on the Docker containers as part of the Product Master image deployment:</p> <ul style="list-style-type: none"> – Red Hat Enterprise Linux (RHEL) 7 Universal Base Image (UBI) base Docker image <p>https://www.ibm.com/docs/en/product-master/12.0.0?topic=deployment-installing-product-by-using-docker-images</p>

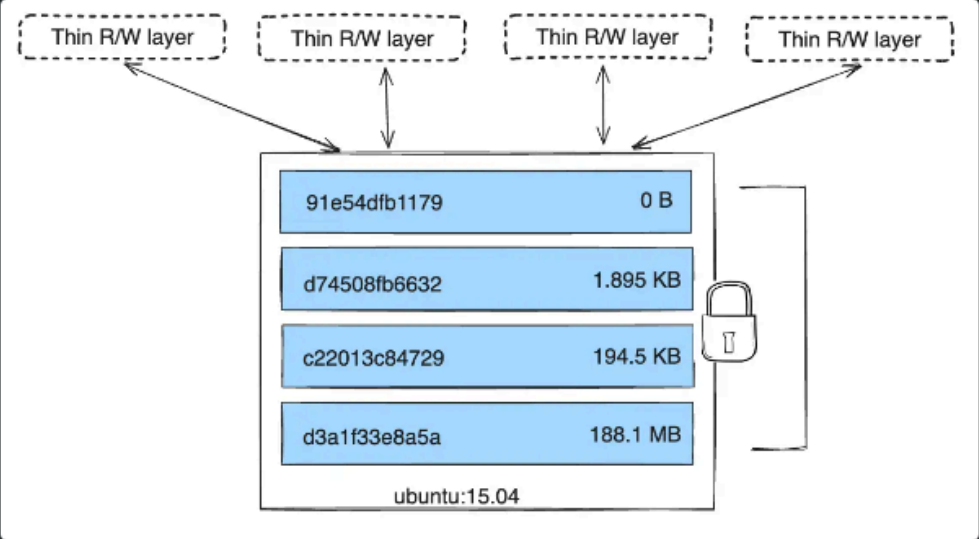
Claim 1	Accused Instrumentalities
	<div><div>ubuntu </div><div>Updated 15 days ago</div><div>Ubuntu is a Debian-based Linux operating system based on free software.</div><div>Linux IBM Z 386 riscv64 x86-64 ARM ARM 64 PowerPC 64 LE</div></div> <div><div>debian </div><div>Updated 35 minutes ago</div><div>Debian is a Linux distribution that's composed entirely of free and open-source software.</div><div>Linux riscv64 x86-64 ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z</div></div> <div>https://hub.docker.com/search?image_filter=official&type=image&q=</div>

Claim 1	Accused Instrumentalities																																																						
	<table><tr><th>Platform</th><th>x86_64 / amd64</th><th>arm64 / aarch64</th><th>arm (32-bit)</th><th>ppc64le</th><th>s390x</th></tr><tr><td>CentOS</td><td>✓</td><td>✓</td><td></td><td>✓</td><td></td></tr><tr><td>Debian</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td></td></tr><tr><td>Fedora</td><td>✓</td><td>✓</td><td></td><td>✓</td><td></td></tr><tr><td>Raspberry Pi OS (32-bit)</td><td></td><td></td><td>✓</td><td></td><td></td></tr><tr><td>RHEL (s390x)</td><td></td><td></td><td></td><td></td><td>✓</td></tr><tr><td>SLES</td><td></td><td></td><td></td><td></td><td>✓</td></tr><tr><td>Ubuntu</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr><tr><td>Binaries</td><td>✓</td><td>✓</td><td>✓</td><td></td><td></td></tr></table> <p>https://docs.docker.com/engine/install/</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p>	Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x	CentOS	✓	✓		✓		Debian	✓	✓	✓	✓		Fedora	✓	✓		✓		Raspberry Pi OS (32-bit)			✓			RHEL (s390x)					✓	SLES					✓	Ubuntu	✓	✓	✓	✓	✓	Binaries	✓	✓	✓		
Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x																																																		
CentOS	✓	✓		✓																																																			
Debian	✓	✓	✓	✓																																																			
Fedora	✓	✓		✓																																																			
Raspberry Pi OS (32-bit)			✓																																																				
RHEL (s390x)					✓																																																		
SLES					✓																																																		
Ubuntu	✓	✓	✓	✓	✓																																																		
Binaries	✓	✓	✓																																																				

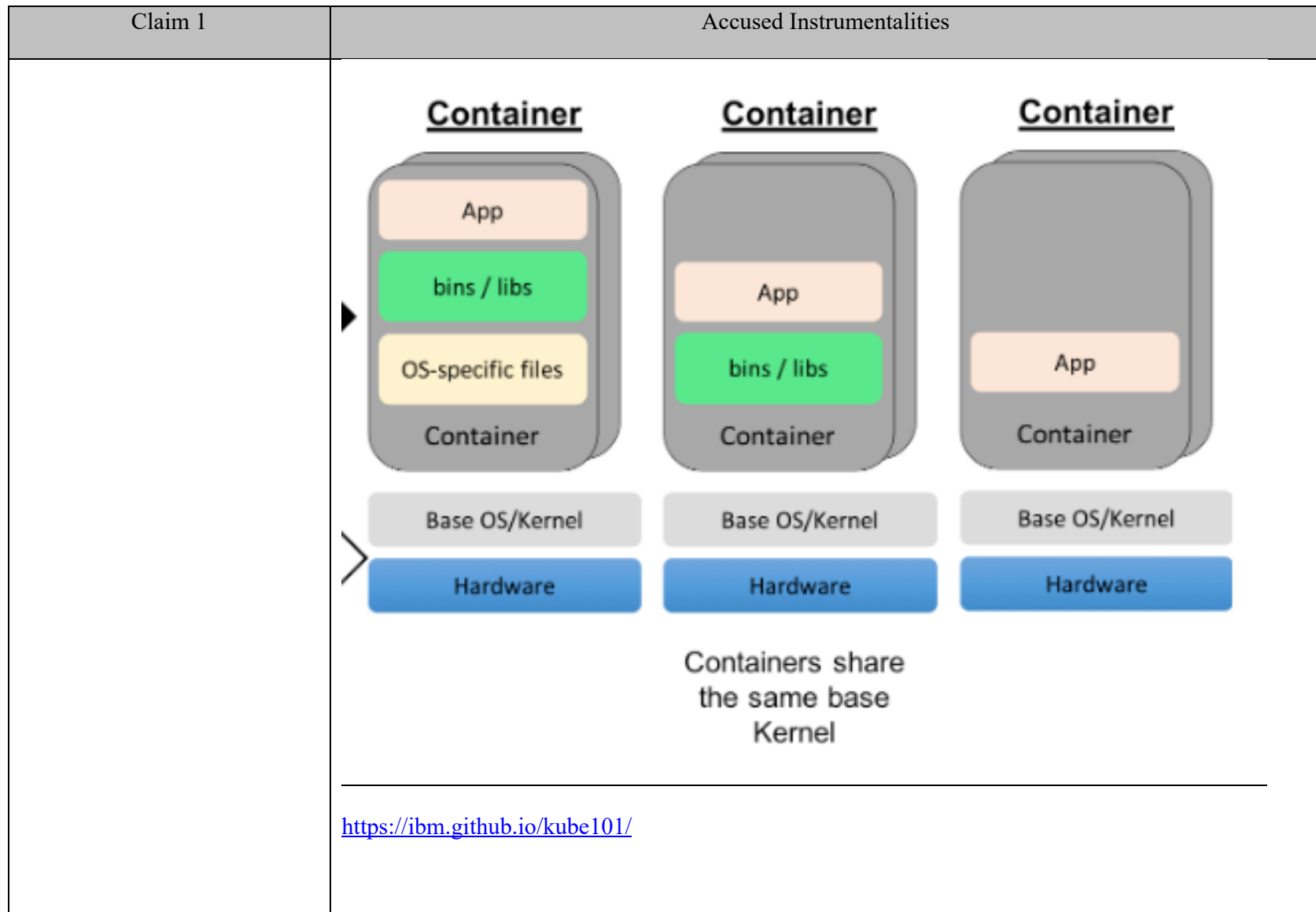
Claim 1	Accused Instrumentalities
	<p>A container is a runtime instance of a docker image.</p> <p>A Docker container consists of</p> <ul style="list-style-type: none">• A Docker image• An execution environment• A standard set of instructions <p>container</p> <p>https://docs.docker.com/glossary/#image</p>

Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>
[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software	In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.

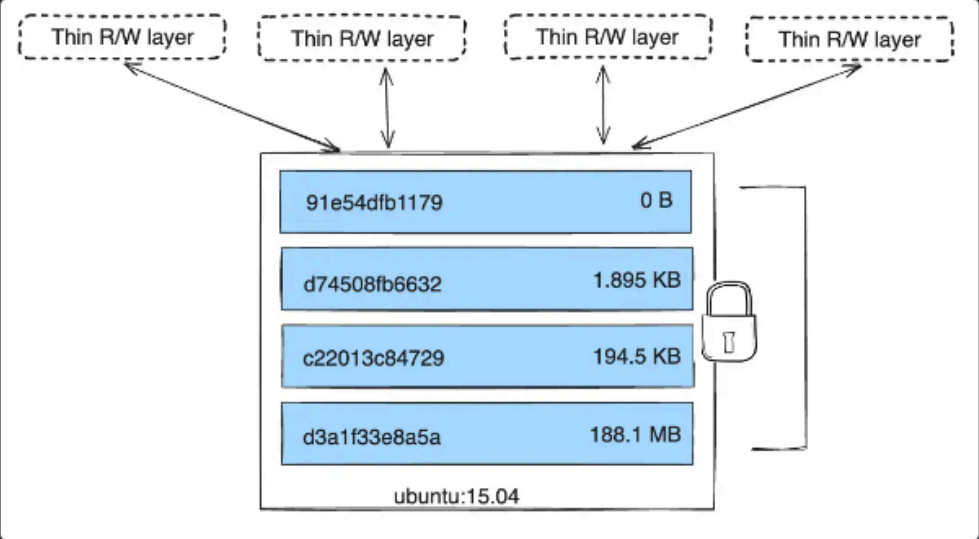
Claim 1	Accused Instrumentalities
<p>applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and</p>	<p>When a Docker image is used to create a container, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications running in their respective containers. The docker image includes essential system files, libraries, and dependencies required to run the software application within the container. The Docker containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it operates in isolation and runs its own instance of the software application without sharing resources or critical system elements with other containers. This ensures that each container has its own isolated context. Docker containers can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same or a different function.</p> <p><i>See, e.g.:</i></p> <p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="674 334 1881 456">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p data-bbox="653 1117 1218 1149">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p>



Claim 1	Accused Instrumentalities
<p>[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p>	<p>In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p> <p>For example, In Docker, each container operates independently, and a Docker base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When a Docker image is used to create a container in ECS, an instance of the SLCSE is provided to a software application. Therefore, different instances of the SLCSE are provided to different applications for performing either a same or a different function, simultaneously.</p> <p><i>See, e.g.:</i></p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="674 334 1881 456">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p> <div data-bbox="793 516 1766 1052"><p>The diagram illustrates the Docker storage architecture. At the top, four dashed boxes labeled 'Thin R/W layer' represent the writable layers for individual containers. These layers are connected by double-headed arrows to a central, shared image stack. The image stack is composed of four layers with the following IDs and sizes: 91e54dfb1179 (0 B), d74508fb6632 (1.895 KB), c22013c84729 (194.5 KB), and d3a1f33e8a5a (188.1 MB). The entire stack is labeled 'ubuntu:15.04' at the bottom. A padlock icon is positioned to the right of the stack, indicating that the underlying image is immutable.</p></div> <p data-bbox="653 1117 1213 1149">https://docs.docker.com/storage/storagedriver/</p> <p data-bbox="659 1192 1753 1330">A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.</p>

Claim 1	Accused Instrumentalities
	https://docs.docker.com/get-started/overview/

Exhibit 5



IBM chooses AppZero as application virtualization partner for SmartCloud

Analyst: Rachel Chalmers

30 Apr, 2012

Since September 2011, IBM Distinguished Engineer Mac Devine has served as the Director and CTO for the SmartCloud portfolio within IBM Global Technology Services (GTS). In practice, that makes him responsible for SmartCloud's technical strategy and architecture, and for choosing SmartCloud partners. Most recently, Devine has been involved in a partnership between AppZero and IBM to provide application migration services to enterprises as they adopt cloud computing.

Company name:

IBM

Activities:

Cloud computing

Head office:

Armonk, NY

Number of employees:

440,885

LY revenue:

\$106.92bn

LY net income:

\$15.86bn

Key suppliers:

AppZero, CohesiveFT, Hadoop

Early Adopter Snapshot

The branded ecosystem of SmartCloud products includes cloud-enabling technologies, PaaS and SaaS, but it's the IaaS offerings that interest us here. Like the club in The Blues Brothers that offers both kinds of music: country and Western, IBM offers both kinds of cloud: public and hybrid. SmartCloud Enterprise (SCE) is a shared public cloud - IBM's answer to Amazon Web Services (AWS) and Rackspace Cloud. For customers that want their cloud private, hosted and managed, IBM offers SmartCloud Enterprise+ (SCE+).

Customers turn to SmartCloud for all kinds of reasons: some to improve their regulatory compliance; others to offer a self-service extension of enterprise infrastructure; still others

are lured by the prospect of a managed service from IBM completely dedicated to them. There are several large financial services firms already up and running on SCE. Typical workloads include development and testing, spillover capacity, early exploration and some big data.

SCE, which has been available since 2011, boasts 2,200 customers. SCE+ has just become generally available. Both are enjoying a steady pipeline of customers who want to transition away from some of Big Blue's other Global Services businesses – for example, e-business hosting, application hosting or strategic outsourcing. Because SmartCloud is more virtualized and automated, it's easier for IBM to run it and cheaper for customers to use.

Strategic vision and business drivers

"All IBM GTS customers are potential SmartCloud users, and that's a multi-billion dollar business," Devine says. "The intent is for us to lower the operations cost while giving them an opportunity to be more flexible in terms of what they're able to accomplish." Where IBM's strategic outsourcing business requires the company to build customized hosted offerings, with SCE+ IBM can offer more standardized services. That said, SmartCloud can't achieve the same level of customization. Everything's a tradeoff.

Nevertheless, cross-selling SmartCloud into the IBM GTS customer base is likely to prove lucrative. But Devine says that besides these cross-selling opportunities, SmartCloud is reaching new customers, as well. Demand is spiking upward. IBM has found that three-quarters of buyers either use or plan to use public cloud services within the next twelve months. There's also a lot of activity around applications and SaaS. Five years ago, Devine observes, VCs funded ISVs to offer licensed software on-premises. Today, the same VCs fund ISVs that sell subscriptions to hosted applications. Devine describes this new breed of software company as 'born in the cloud.'

Challenges and obstacles

These apps don't always stay in the cloud. IBM sees apps developed in hosted environments being 'harvested' and brought on-premises. It also sees organizations looking to migrate enterprise

applications into the public cloud. Because of this fundamental need for two-way freedom of movement, Devine says, mobility is key. (This hybrid pattern also explains IBM's decision to offer both SCE and SCE+.)

To Devine, a cloud-enabled workload is one that was born in the enterprise but needs to be able to run either on-premises or in the public cloud. Then there are cloud-centric workloads, typically agile extensions to existing assets or all-new applications. With SmartCloud, IBM is trying to address both workloads. That's why it has turned to AppZero and to Cohesive Flexible Technologies (CohesiveFT) to give its prospects and customers comprehensive application migration capabilities.

Think of it this way. There are two kinds of workloads – cloud-enabled (born in the enterprise) and cloud-centric (born on the cloud) – that IBM wishes to help its customers move into two kinds of cloud: SCE and SCE+. With AppZero, IBM can capture an application environment, transform it and migrate it to a SmartCloud environment. CohesiveFT performs the same kind of encapsulation at the network-virtualization layer.

Deployment summary

It was Devine who performed the due diligence exercise in 2011 that reviewed IBM's options for migration tools. He narrowed down the options to CohesiveFT and AppZero, and then he introduced the companies to each other. Through the resulting Elastic Enterprise Applications partnership, Cohesive handles networking and automation for Linux, while AppZero offers a Windows virtualized application container. Devine is proud that the partners got an offering up and operational in six weeks.

His problem since has been figuring out how best to wrap IBM's engine around that offering. He turned to an application migration services team that already existed within GTS. In the past, when a customer wanted to move from their on-premises environment to, say, strategic outsourcing, this team would assemble various tools and manual processes to make that migration happen. Now, the team can turn to AppZero and CohesiveFT for rapid migration to the far more standardized set of features and functions embodied in SCE+. Customers that can do it themselves may take advantage of self-service migration to SCE. Either way, ideally, the customers should end up with a more optimized environment at a lower cost point.

Innovation and roadmap

AppZero is not without its limitations. It can move like OS to like OS – say, Windows 2003 to Windows 2003, or Windows 2008 R2 to Windows 2008 R2 – but it can't move Windows to Linux. The

software has to stop the application before moving it. Moving shared databases will move all databases unless a DBA manually prunes the migration. It doesn't support SharePoint or Exchange.

There are some powerful upsides, though. It's fast – migration takes minutes rather than hours – and it's great for migrating (for example) HP OpenView users across to IBM Tivoli. There's no vendor lock-in because the container dissolves after the move. And while it's constrained by OS, AppZero can migrate to and from any hypervisor, physical or cloud environment. This is a key part of its appeal for IBM because SmartCloud is based on KVM and Power virtualization, and the prospective customers mostly use VMware. In other words, AppZero is strategically critical to IBM's ability to extract workloads from its customers' proprietary environments and run them on IBM's own relatively open cloud. That's a win for customers and perhaps an even bigger win for IBM.

Reproduced by permission of 451 Research; © 2011. This report was originally published within 451 Research's Market Insight Service. For additional information on 451 Research or to apply for trial access, go to: www.451research.com

EXHIBIT B

**UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

VIRTAMOVE, CORP.,

Plaintiff,

v.

HEWLETT PACKARD ENTERPRISE
COMPANY,

Defendant.

Case No. 2:24-cv-00093-JRG

(LEAD CASE)

JURY TRIAL DEMANDED

**SECOND AMENDED COMPLAINT FOR PATENT INFRINGEMENT AGAINST
HEWLETT PACKARD ENTERPRISE COMPANY**

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, “Plaintiff” or “VirtaMove”) makes the following allegations against Defendant Hewlett Packard Enterprise Company (collectively, “Defendant” or “HPE”):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant’s unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058 (collectively, the “Asserted Patents”). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as Appzero Software Corp. (“Appzero”), which was established in 2010.

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repackage, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove's mission is to move and modernize the world's server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach \$2.1 billion in 2019 and \$4.3 billion in 2022—a compound annual growth rate (“CAGR”) of 30%. *See, e.g.*, <https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022>. Another source reported the application containers market had a market size of \$5.45 billion in 2024 and estimated it to reach \$19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.*, <https://www.mordorintelligence.com/industry-reports/application-container-market>.

6. Defendant Hewlett Packard Enterprise Company is a corporation organized under the laws of the State of Delaware, with its principal place of business at 1701 E Mossy Oaks Road, Spring, Texas 77389. Hewlett Packard Enterprise Company has as its registered agent for service: C T Corporation System, 1999 Bryan St., Ste. 900, Dallas, Texas 75201-3136. Hewlett Packard

Enterprise Company has been registered to do business in the state of Texas under Texas SOS file number 802175187.

JURISDICTION AND VENUE

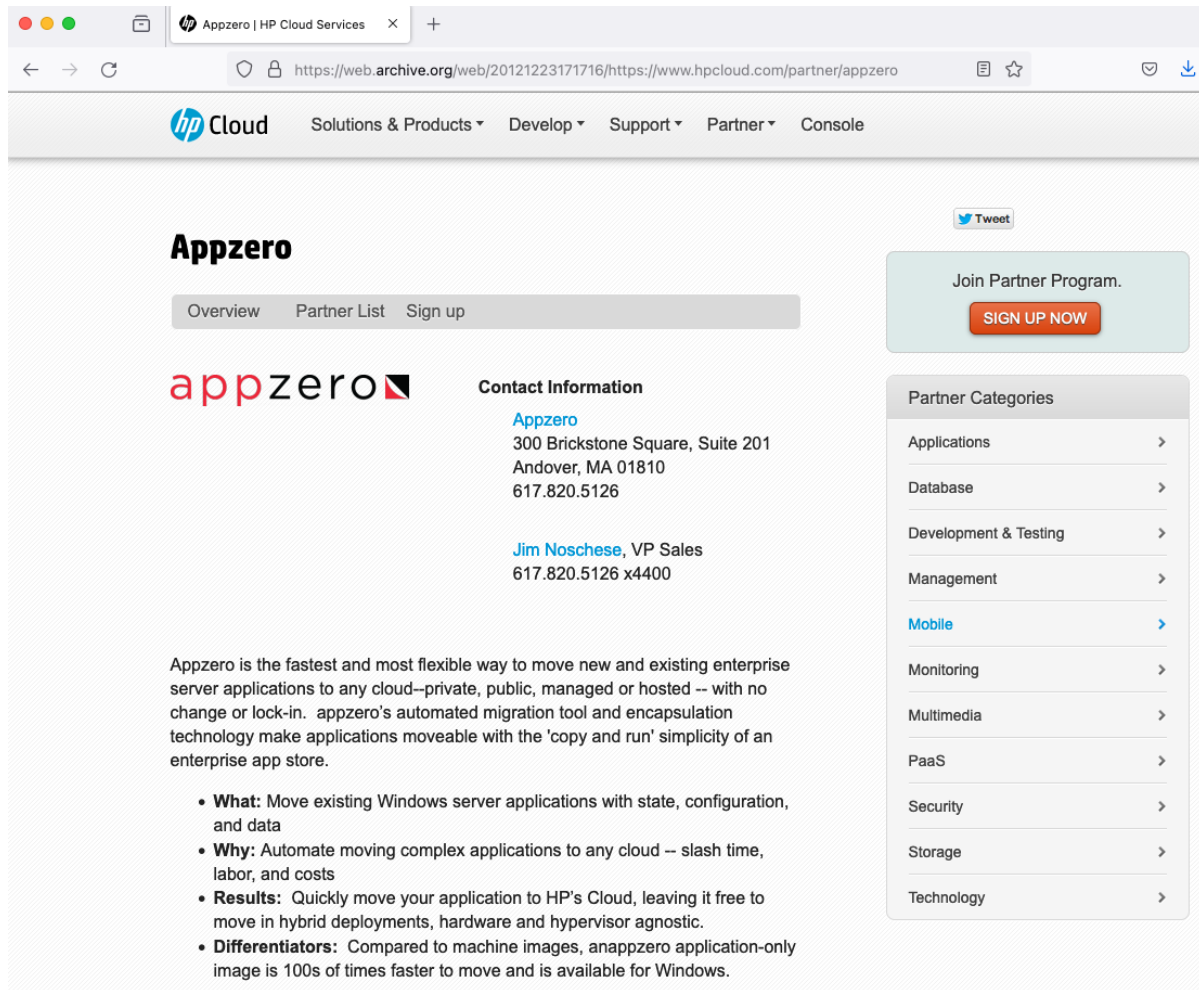
7. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

9. Venue is proper in this District. For example, HPE has a regular and established place of business, including, e.g., at 6080 Tennyson Parkway, Suite 400, Plano, Texas 75024.

ADDITIONAL FACTS

10. As shown by the WayBack Machine, HP Co. and Appzero were part of a “partner” program as far back as 2012. HP Co. would later split into HPE and HP Inc. in 2014.



11. In meetings between 2011–2019, representatives of VirtaMove met with representatives of HP Co. and HPE to discuss and demo AppZero and its technology for purposes of investment, use, sale, and/or partnership. During this time frame, HPE would have learned that AppZero was patented as a matter of basic due diligence, or HPE would have been willfully blind to this fact. After the demos and disclosures from VirtaMove, the patented AppZero technology has made its way into the Accused Products of HPE described below.

COUNT I

INFRINGEMENT OF U.S. PATENT NO. 7,519,814

12. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if

fully set forth herein.

13. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 ('814 patent), titled "System for Containerization of Application Sets," issued on April 14, 2009. A true and correct copy of the '814 Patent is attached as Exhibit 1.

14. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., HPE's Ezmeral Runtime Enterprise, that directly infringe, literally and/or under the doctrine of equivalents, claims of the '814 patent, for example:

HPE Ezmeral Runtime Enterprise is an enterprise-grade container orchestration platform that is designed for the containerization of both cloud-native and non-cloud-native monolithic applications with persistent data. It deploys 100% open-source Kubernetes for orchestration, provides a state-of-the-art file system and data fabric for persistent container storage, and provides enterprises with the ability to deploy non-cloud-native AI and Analytics workloads in containers. Enterprises can now easily extend the agility and efficiency benefits of containers to more of their enterprise applications—running on either bare-metal or virtualized infrastructure, on-premises, in multiple clouds, or at the edge.

https://www.hpe.com/psnow/doc/a50004264enw.pdf?jumpid=in_pdp-psnow-qs.

15. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

16. Defendant's infringement has been and is willful. Defendant knew of VirtaMove, its products, and at least one of the patents long before this suit was filed and at least as early as 2012. For example, the '814 Patent was cited in connection with at least the following patents: U.S. Patent Nos. 10,489,354, 10,749,740, 11,467,775, 11,687,267, 11,693,573. Defendant knew, or should have known, that its conduct amounted to infringement of the '814 patent. Accordingly, Defendant is liable for willful infringement.

17. Defendant also knowingly and intentionally induces infringement of claims of the

'814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and/or earlier, as set forth above. Despite this knowledge of the '814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '814 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '814 patent, thereby specifically intending for and inducing its customers to infringe the '814 patent through the customers' normal and customary use of the Accused Products.

18. Defendant has also infringed, and continue to infringe, claims of the '814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '814 patent, in violation of 35 U.S.C. § 271(c).

19. The Accused Products satisfy all claim limitations of claims of the '814 patent. A claim chart comparing an independent claim of the '814 patent to a representative Accused Product, is attached as Exhibit 2, which is hereby incorporated by reference in its entirety.

20. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the '814

patent pursuant to 35 U.S.C. § 271.

21. As a result of Defendant's infringement of the '814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. VirtaMove has complied with the requirements of 35 U.S.C. § 287 and is not aware of any unmarked products that practice the claims of the '814 patent. In the alternative, either VirtaMove's product was marked before the filing of this lawsuit, or no requirement for marking applies.

COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

22. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

23. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 ('058 patent), titled "Computing System Having User Mode Critical System Elements as Shared Libraries," issued on August 24, 2010. A true and correct copy of the '058 patent is attached as Exhibit 3.

24. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., HPE's Ezmeral Runtime Enterprise, that directly infringe, literally and/or under the doctrine of equivalents, claims of the '058 patent, for example:

HPE Ezmeral Runtime Enterprise is a unified platform built on open-source Kubernetes and designed for both cloud-native applications and non-cloud-native applications running on any infrastructure; whether on-premises, in multiple public clouds, in a hybrid model, or at the edge.

https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&page=home/about-hpe-ezmeral-container-pl/Welcome.html.

25. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

26. Defendant also knowingly and intentionally induces infringement of claims of the '058 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed. Despite this knowledge of the '058 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '058 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '058 patent, thereby specifically intending for and inducing its customers to infringe the '058 patent through the customers' normal and customary use of the Accused Products.

27. Defendant has also infringed, and continue to infringe, claims of the '058 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '058 patent, in

violation of 35 U.S.C. § 271(c).

28. The Accused Products satisfy all claim limitations of claims of the '058 patent. A claim chart comparing an independent claim of the '058 patent to a representative Accused Product, is attached as Exhibit 4, which is hereby incorporated by reference in its entirety.

29. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the '058 patent pursuant to 35 U.S.C. § 271.

30. As a result of Defendant's infringement of the '058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. VirtaMove has complied with the requirements of 35 U.S.C. § 287 and is not aware of any unmarked products that practice the claims of the '058 patent. In the alternative, either VirtaMove's product was marked before the filing of this lawsuit, or no requirement for marking applies.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

- a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;
- b. A judgment in favor of Plaintiff that Defendant has willfully infringed the '814 patent;
- c. A permanent injunction prohibiting Defendant from further acts of infringement of each of the '814 and '058 patents;

- d. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant's infringement of each of the Asserted Patents;
- e. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;
- f. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys' fees against Defendant; and
- g. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury of any issues so triable by right.

Dated: June 24, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)

rmirzaie@raklaw.com

Marc A. Fenster (CA SBN 181067)

mfenster@raklaw.com

Neil A. Rubin (CA SBN 250761)

nrubin@raklaw.com

Amy E. Hayden (CA SBN 287026)

ahayden@raklaw.com

Jacob R. Buczko (CA SBN 269408)

jbuczko@raklaw.com

James S. Tsuei (CA SBN 285530)

jtsuei@raklaw.com

James A. Milkey (CA SBN 281283)

jmilkey@raklaw.com

Christian W. Conkle (CA SBN 306374)

cconkle@raklaw.com

Jonathan Ma (CA SBN 312773)

jma@raklaw.com

Daniel Kolko (CA SBN 341680)

dkolko@raklaw.com

RUSS AUGUST & KABAT

12424 Wilshire Boulevard, 12th Floor

Los Angeles, CA 90025

Telephone: (310) 826-7474

Qi (Peter) Tong (TX SBN 24119042)

ptong@raklaw.com

RUSS AUGUST & KABAT

4925 Greenville Ave., Suite 200

Dallas, TX 75206

Telephone: (310) 826-7474

Attorneys for Plaintiff VirtaMove, Corp.

CERTIFICATE OF SERVICE

I certify that on June 24, 2024 a true and correct copy of the foregoing document was electronically filed with the Court and served on all parties of record via the Court's CM/ECF system.

/s/ Reza Mirzaie

Reza Mirzaie

CERTIFICATE OF CONFERENCE

I certify that on June 24, 2024, I conferred with counsel for HPE, who confirmed that HPE is not opposed to the filing of this Second Amended Complaint under Fed. R. Civ. P. 15(a)(2), which provides, in part, that "a party may amend its pleading . . . with the opposing party's written consent"

/s/ Qi (Peter) Tong

Qi (Peter) Tong

Exhibit 1



US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,519,814 B2**
(45) **Date of Patent:** **Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

WO WO 2006/039181 A 4/2006

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248;
709/214; 719/319

(58) **Field of Classification Search** 713/167;
719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

OTHER PUBLICATIONS

Soltész, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667.*

Primary Examiner—Kambiz Zand

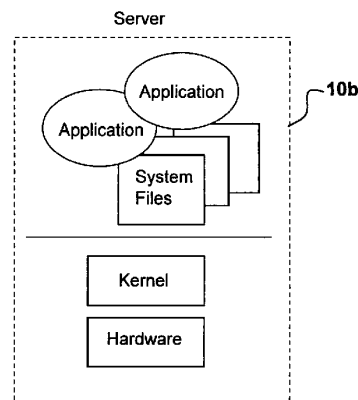
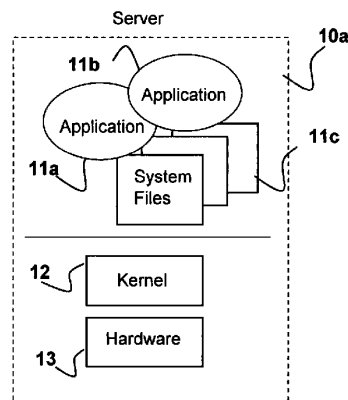
Assistant Examiner—Ronald Baum

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



US 7,519,814 B2

Page 2

U.S. PATENT DOCUMENTS				2002/0004854 A1	1/2002	Hartley	
6,847,970 B2 *	1/2005	Keller et al.	707/100	2002/0174215 A1	11/2002	Schaefer	709/224
7,076,784 B1 *	7/2006	Russell et al.	719/315	2003/0101292 A1	5/2003	Fisher	
7,287,259 B2 *	10/2007	Grier et al.	719/331	* cited by examiner			

U.S. Patent

Apr. 14, 2009

Sheet 1 of 17

US 7,519,814 B2

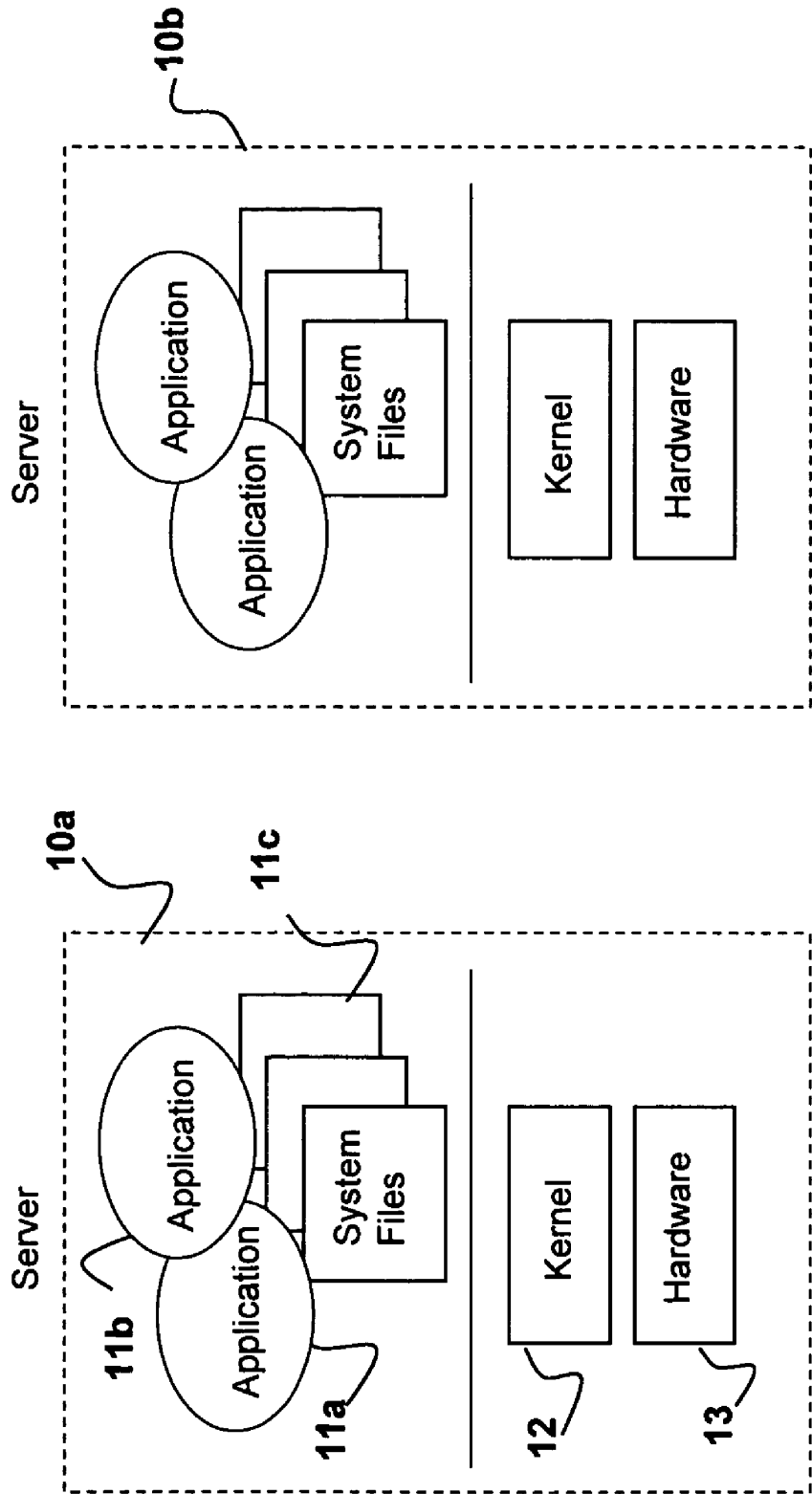


Figure 1

U.S. Patent

Apr. 14, 2009

Sheet 2 of 17

US 7,519,814 B2

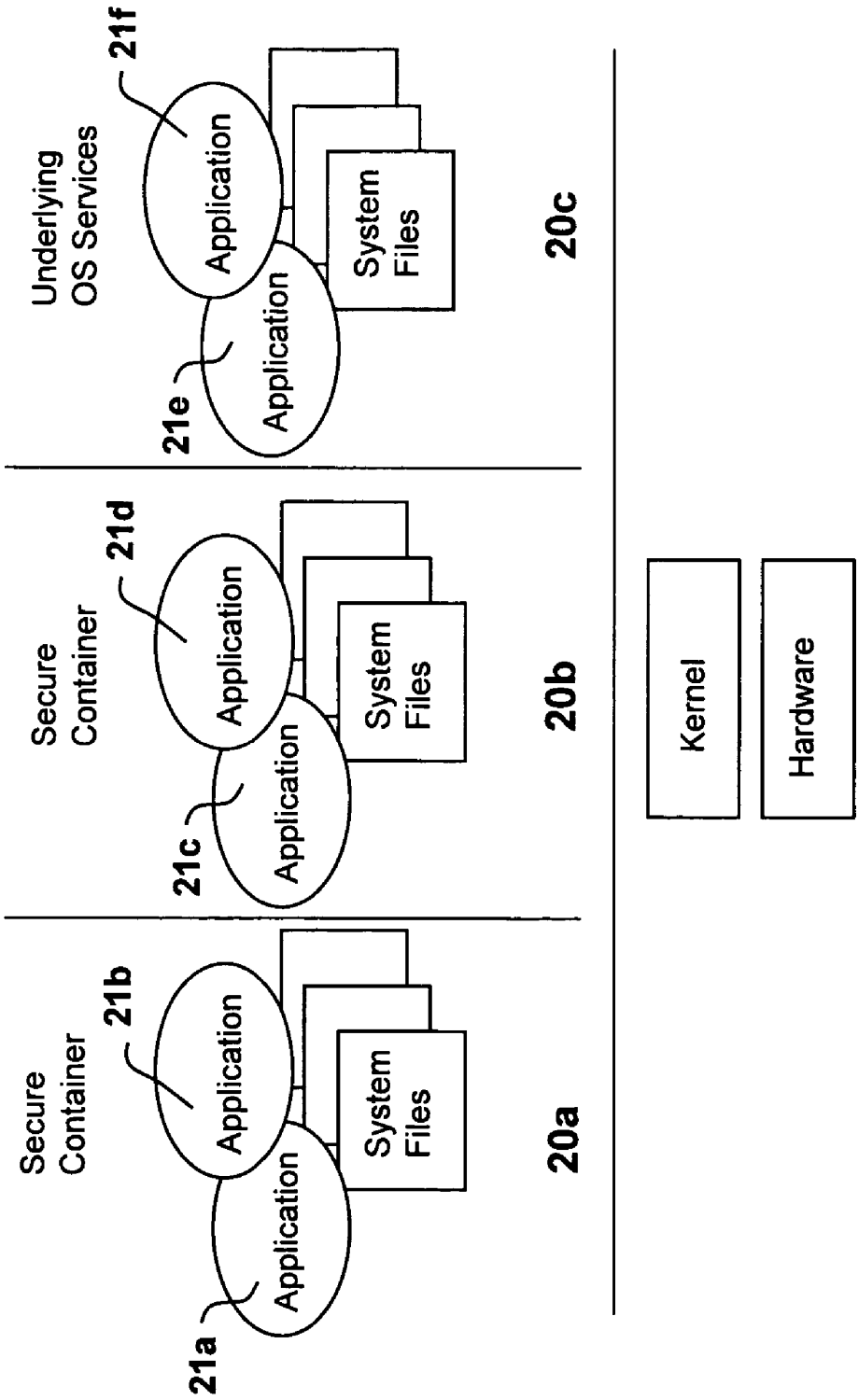


Figure 2

U.S. Patent

Apr. 14, 2009

Sheet 3 of 17

US 7,519,814 B2

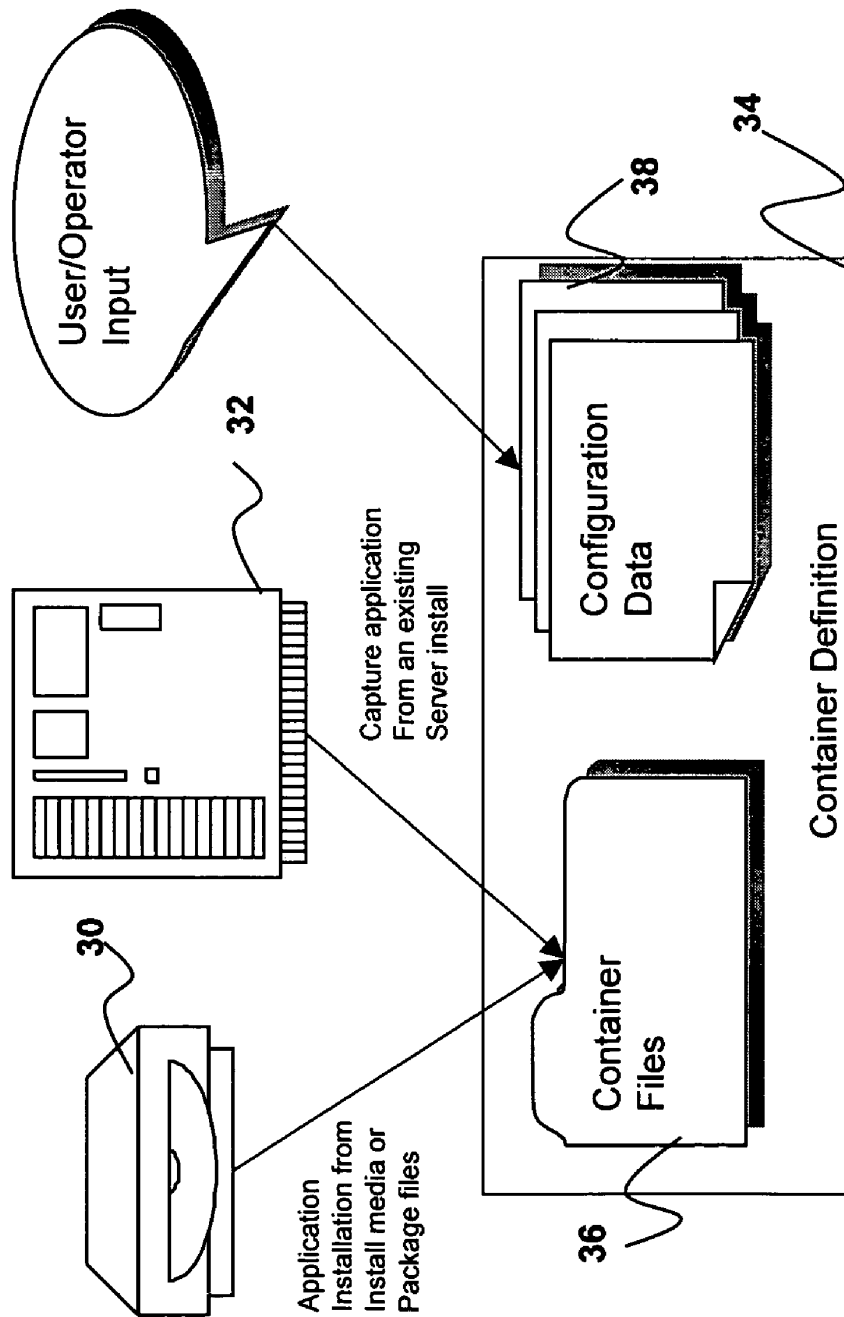


Figure 3

U.S. Patent

Apr. 14, 2009

Sheet 4 of 17

US 7,519,814 B2

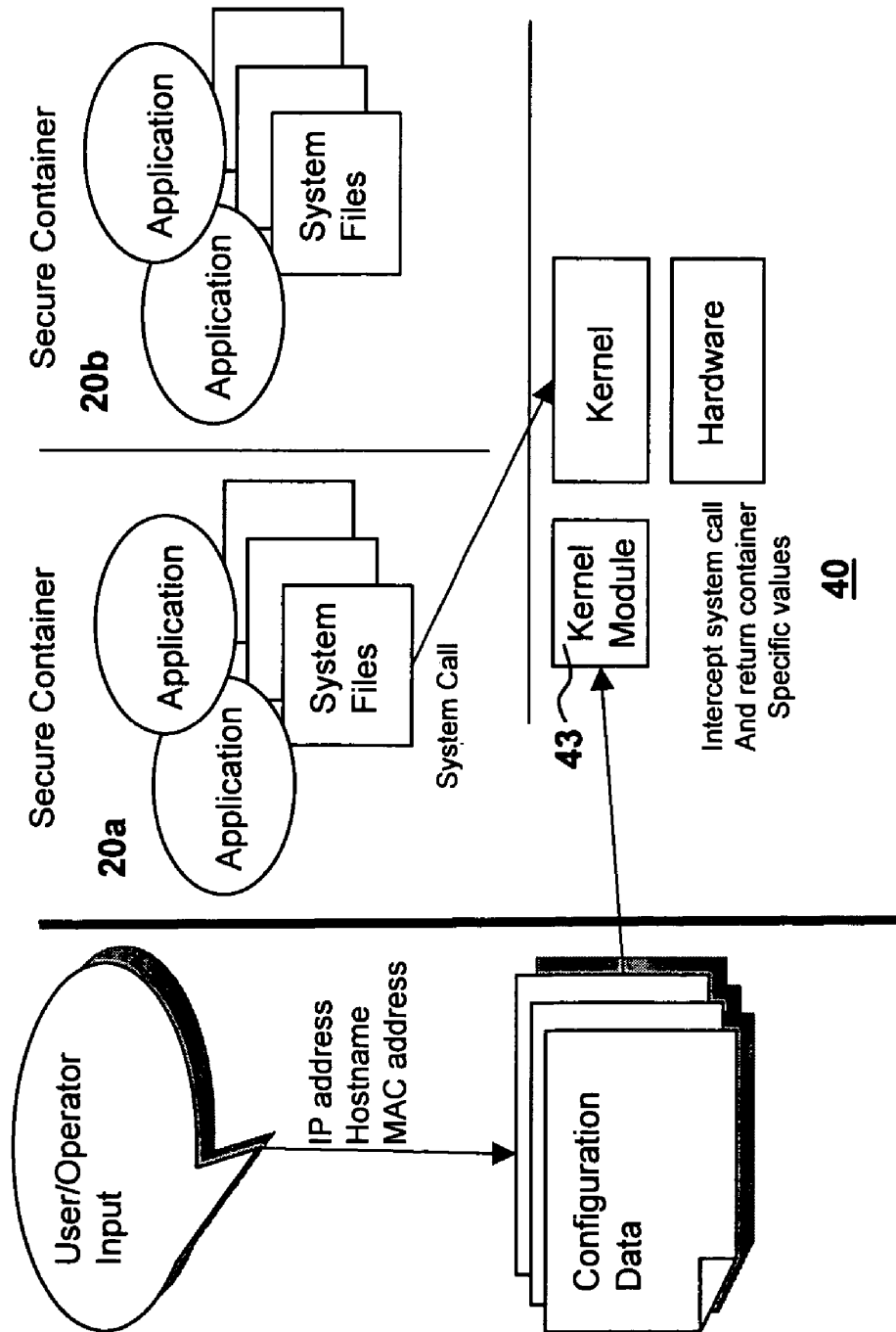


Figure 4

U.S. Patent

Apr. 14, 2009

Sheet 5 of 17

US 7,519,814 B2

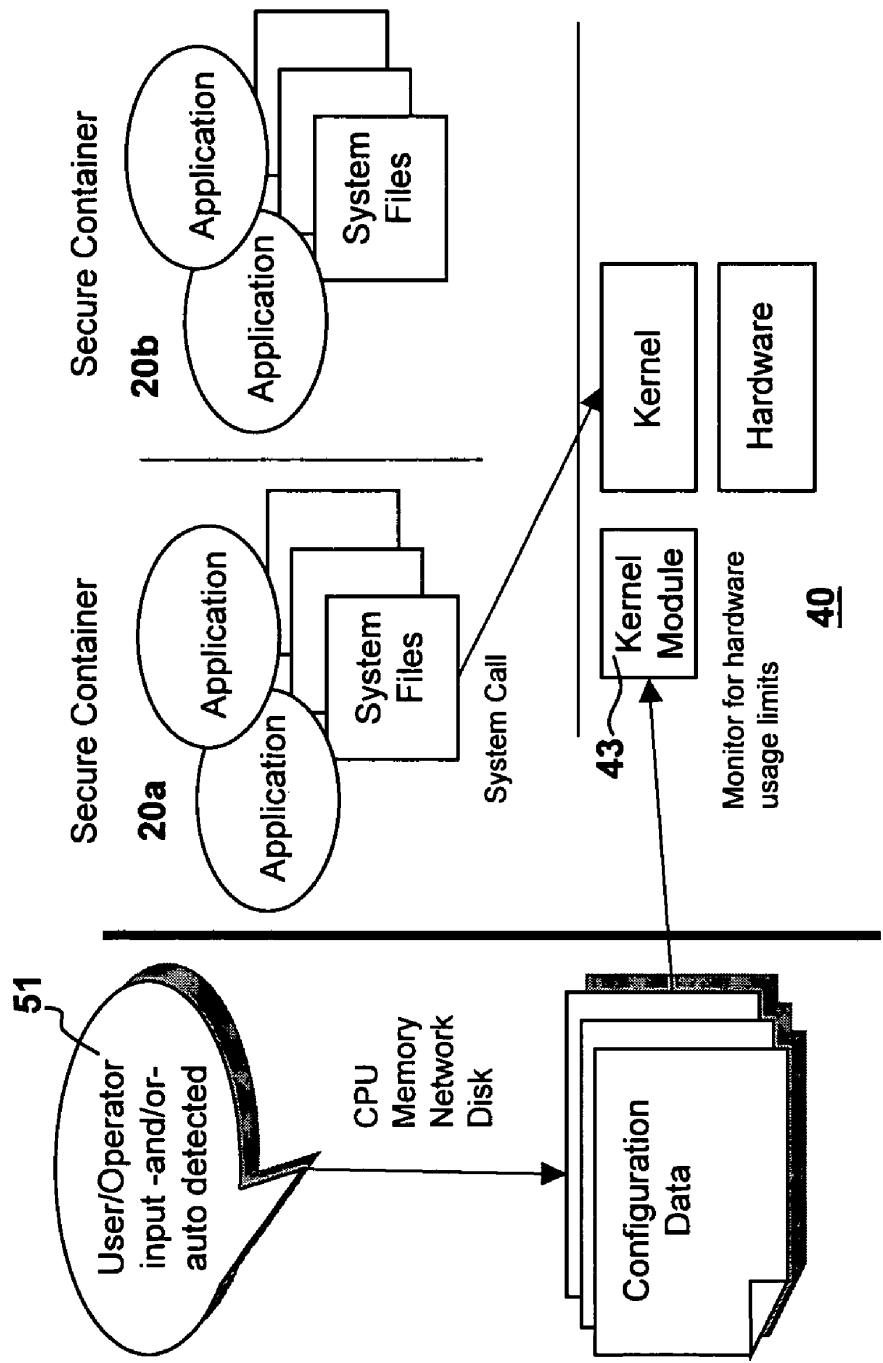


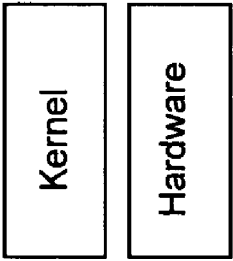
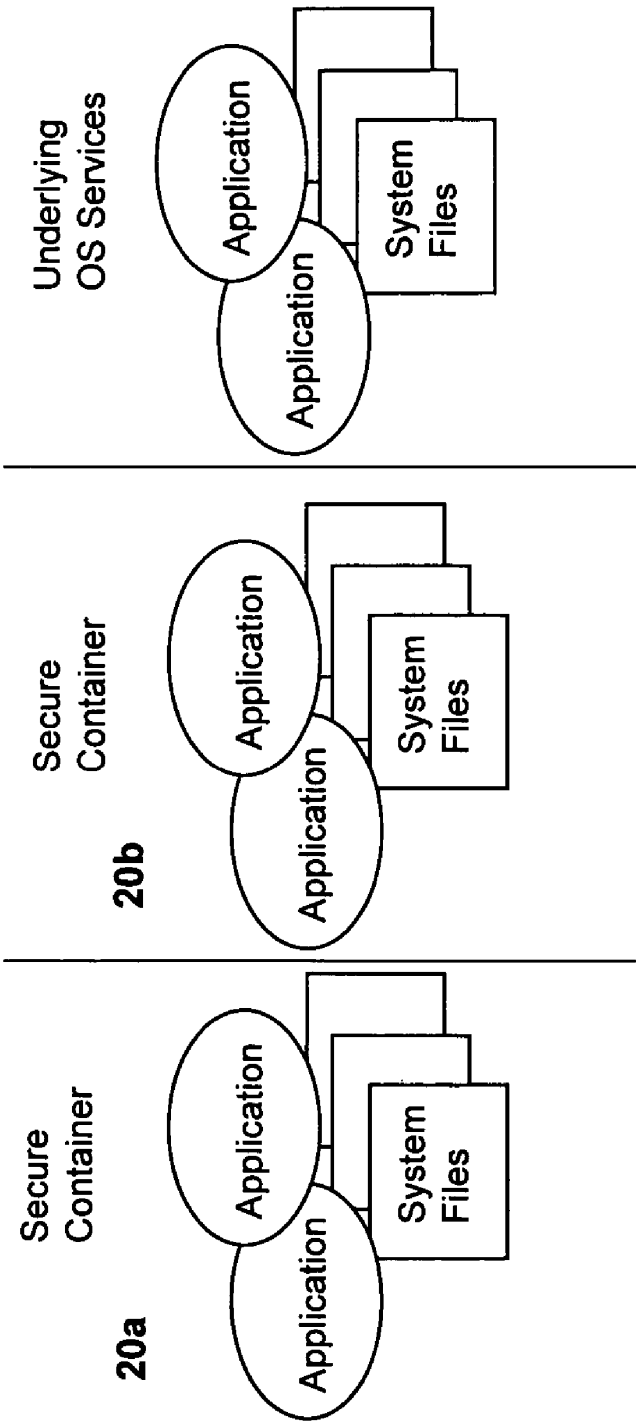
Figure 5

U.S. Patent

Apr. 14, 2009

Sheet 6 of 17

US 7,519,814 B2



40

Figure 6

U.S. Patent

Apr. 14, 2009

Sheet 7 of 17

US 7,519,814 B2

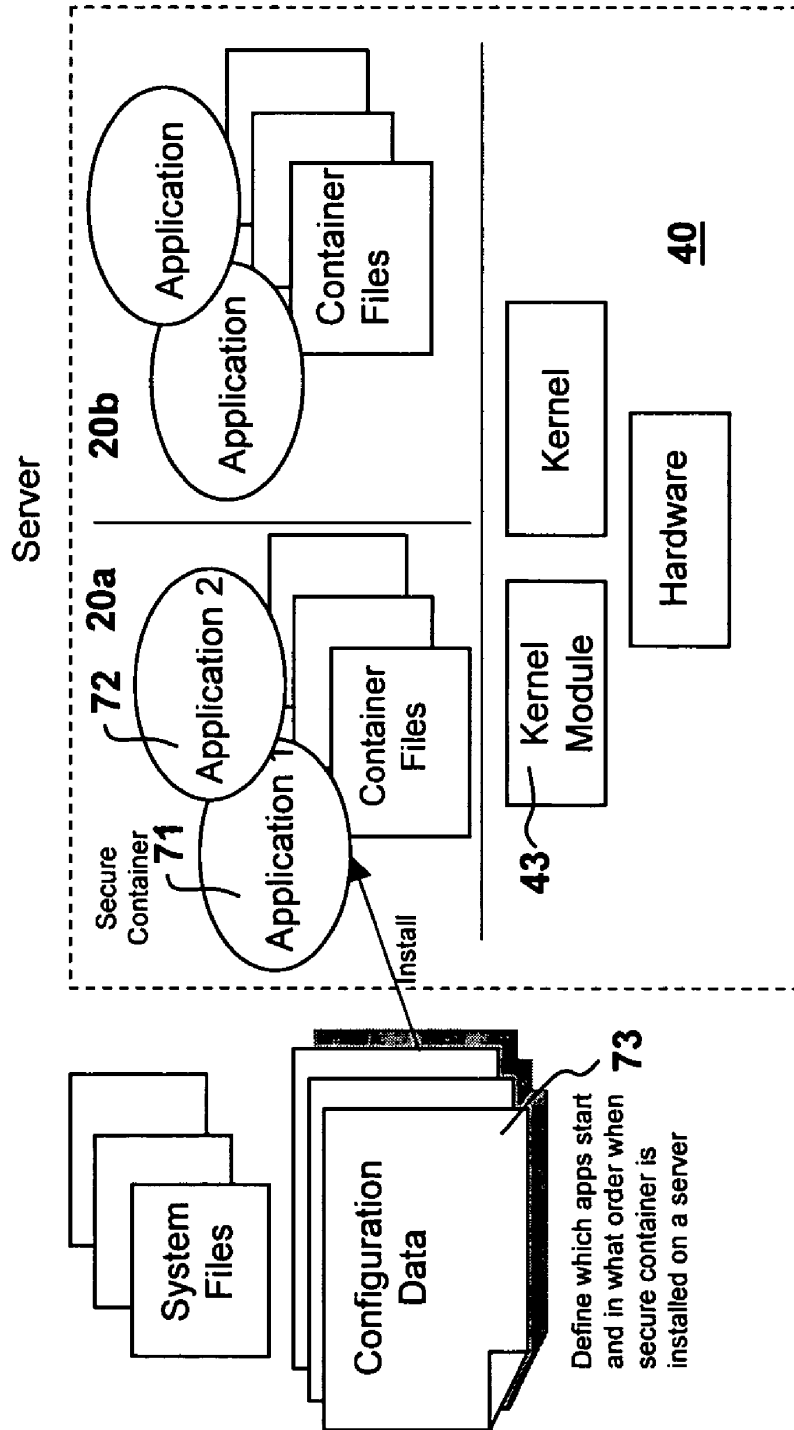


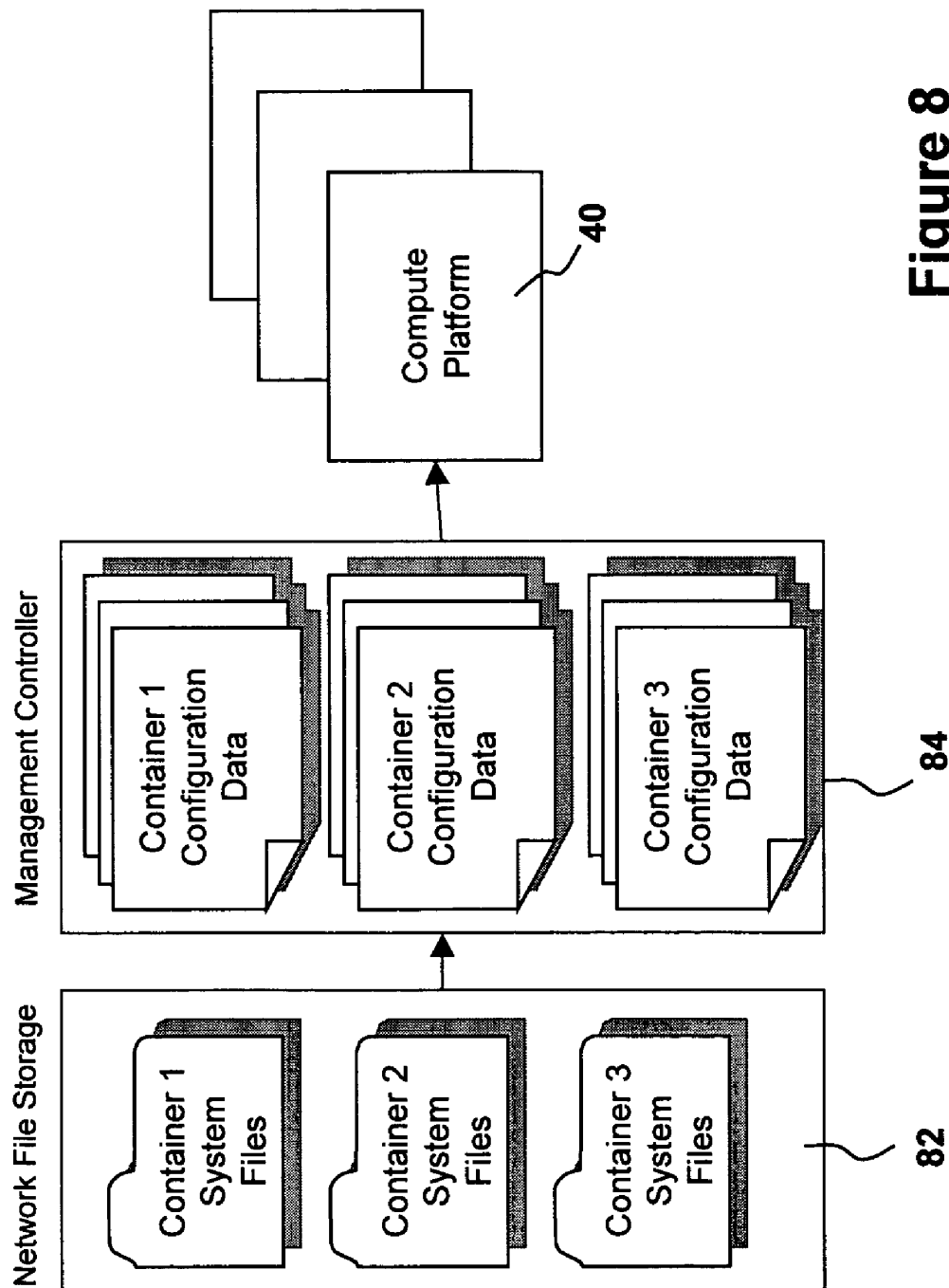
Figure 7

U.S. Patent

Apr. 14, 2009

Sheet 8 of 17

US 7,519,814 B2

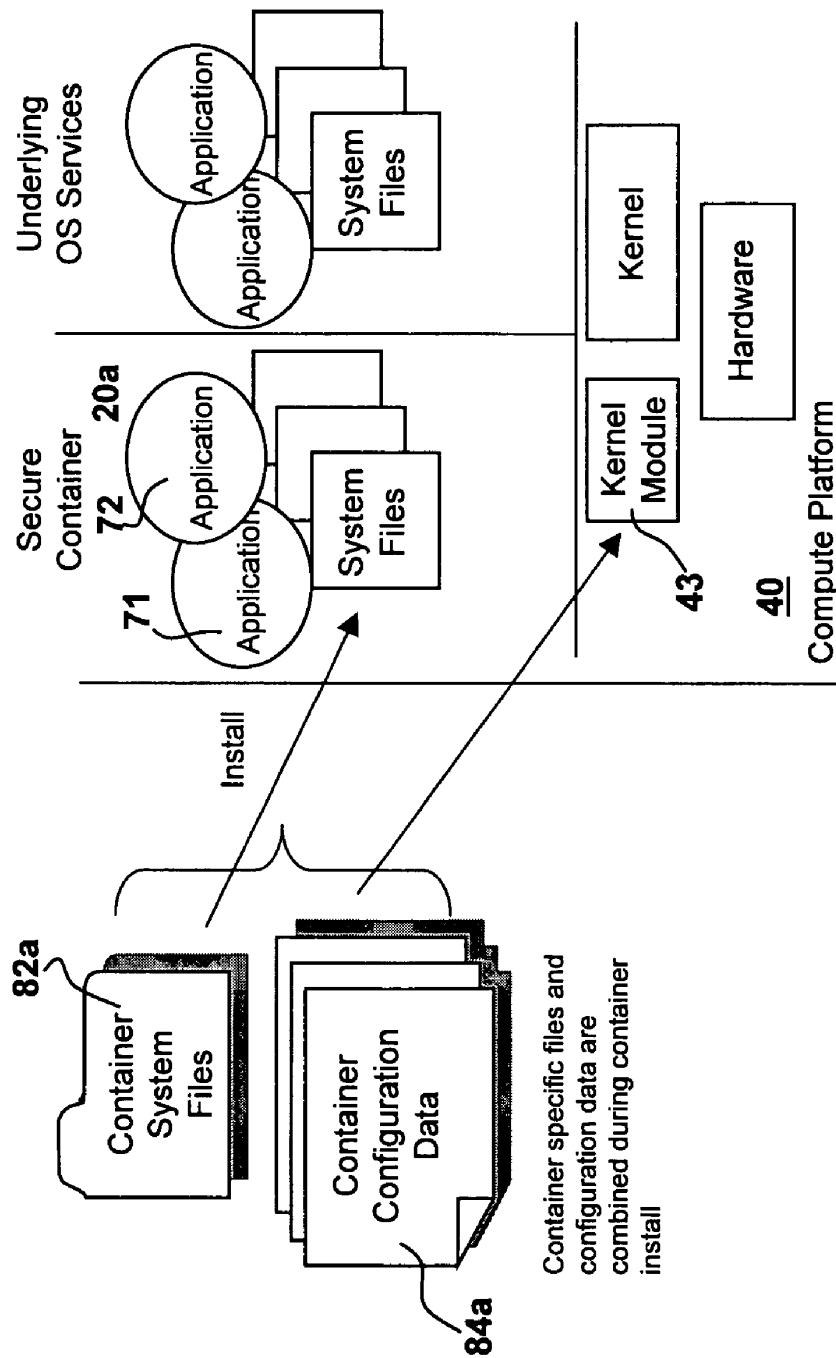


U.S. Patent

Apr. 14, 2009

Sheet 9 of 17

US 7,519,814 B2

**Figure 9**

U.S. Patent

Apr. 14, 2009

Sheet 10 of 17

US 7,519,814 B2

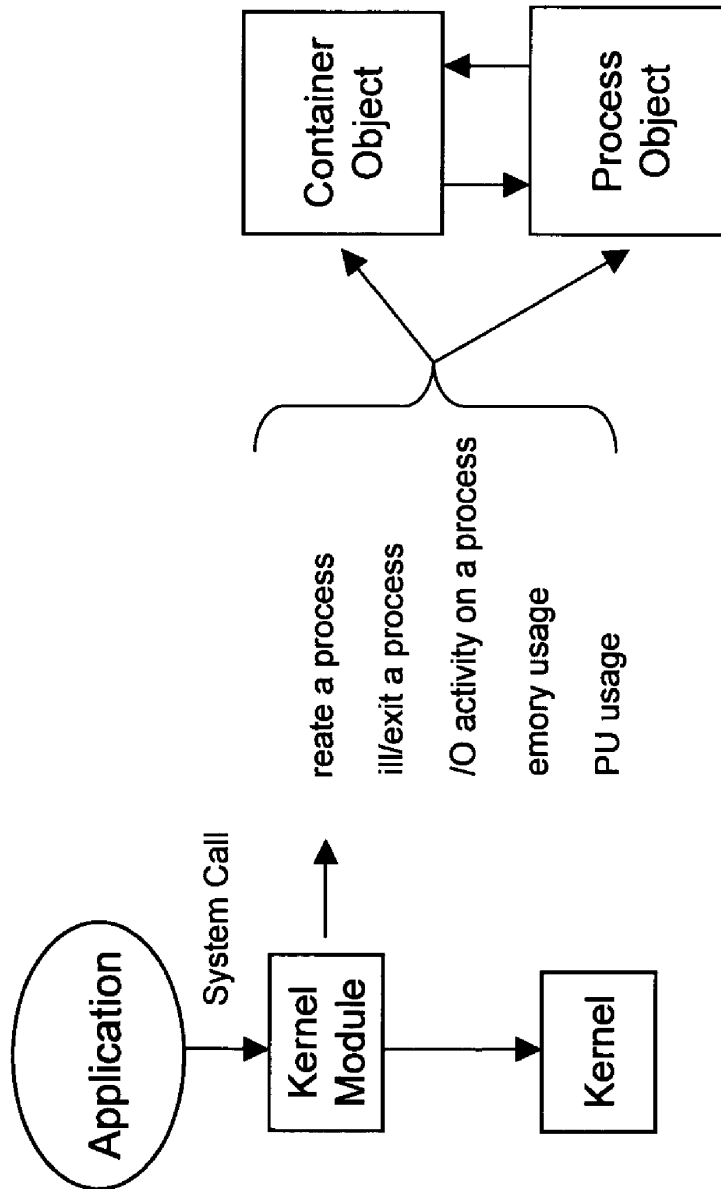


Figure 10

U.S. Patent

Apr. 14, 2009

Sheet 11 of 17

US 7,519,814 B2

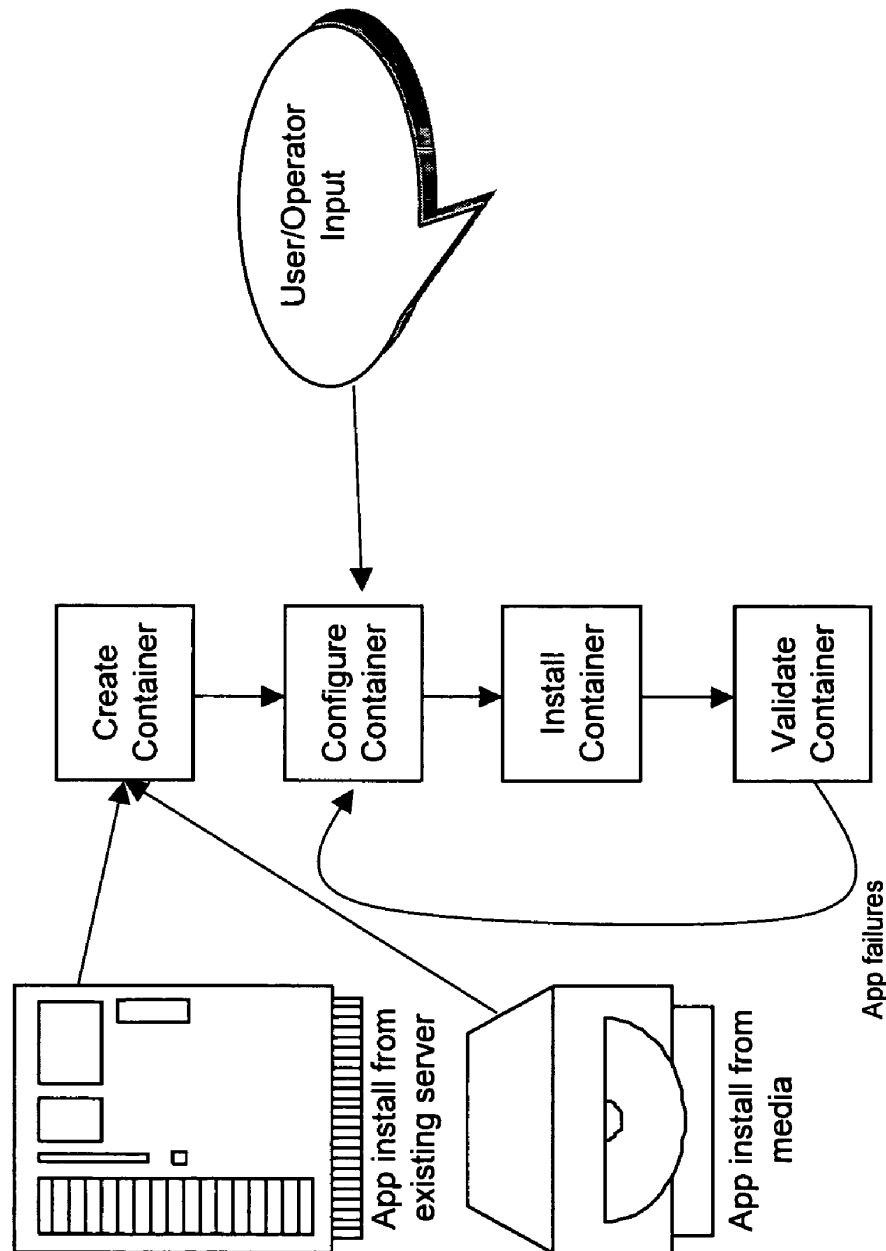


Figure 11

U.S. Patent

Apr. 14, 2009

Sheet 12 of 17

US 7,519,814 B2

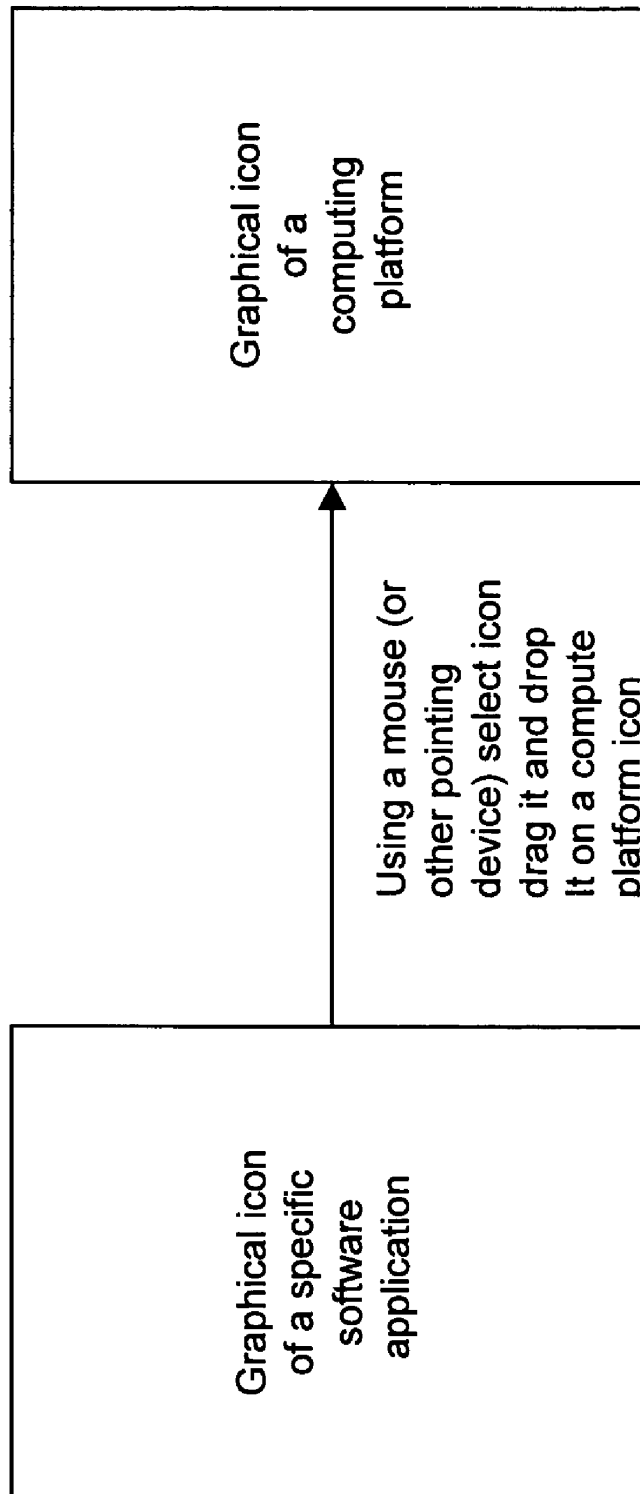


Figure 12

U.S. Patent

Apr. 14, 2009

Sheet 13 of 17

US 7,519,814 B2

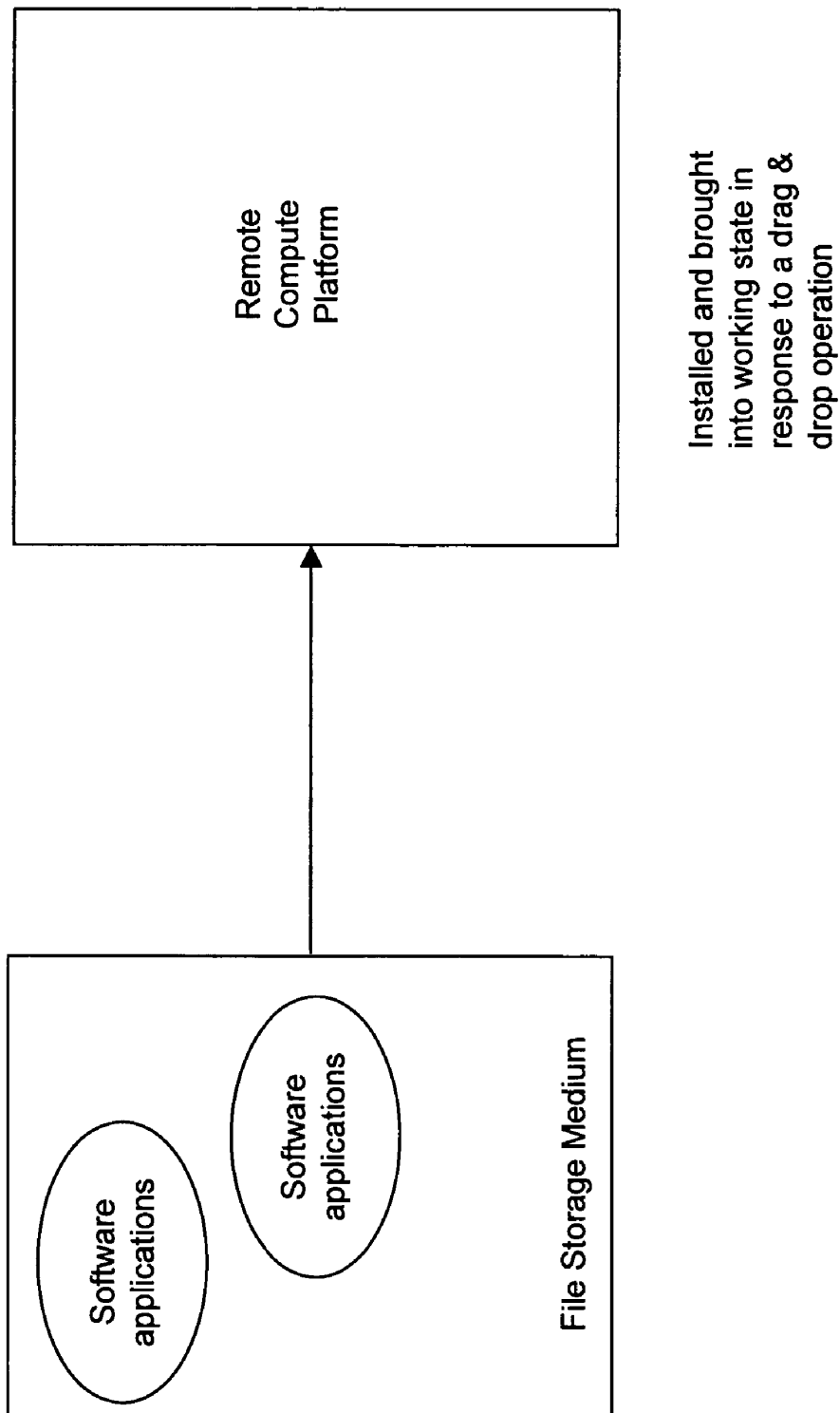


Figure 13

U.S. Patent

Apr. 14, 2009

Sheet 14 of 17

US 7,519,814 B2

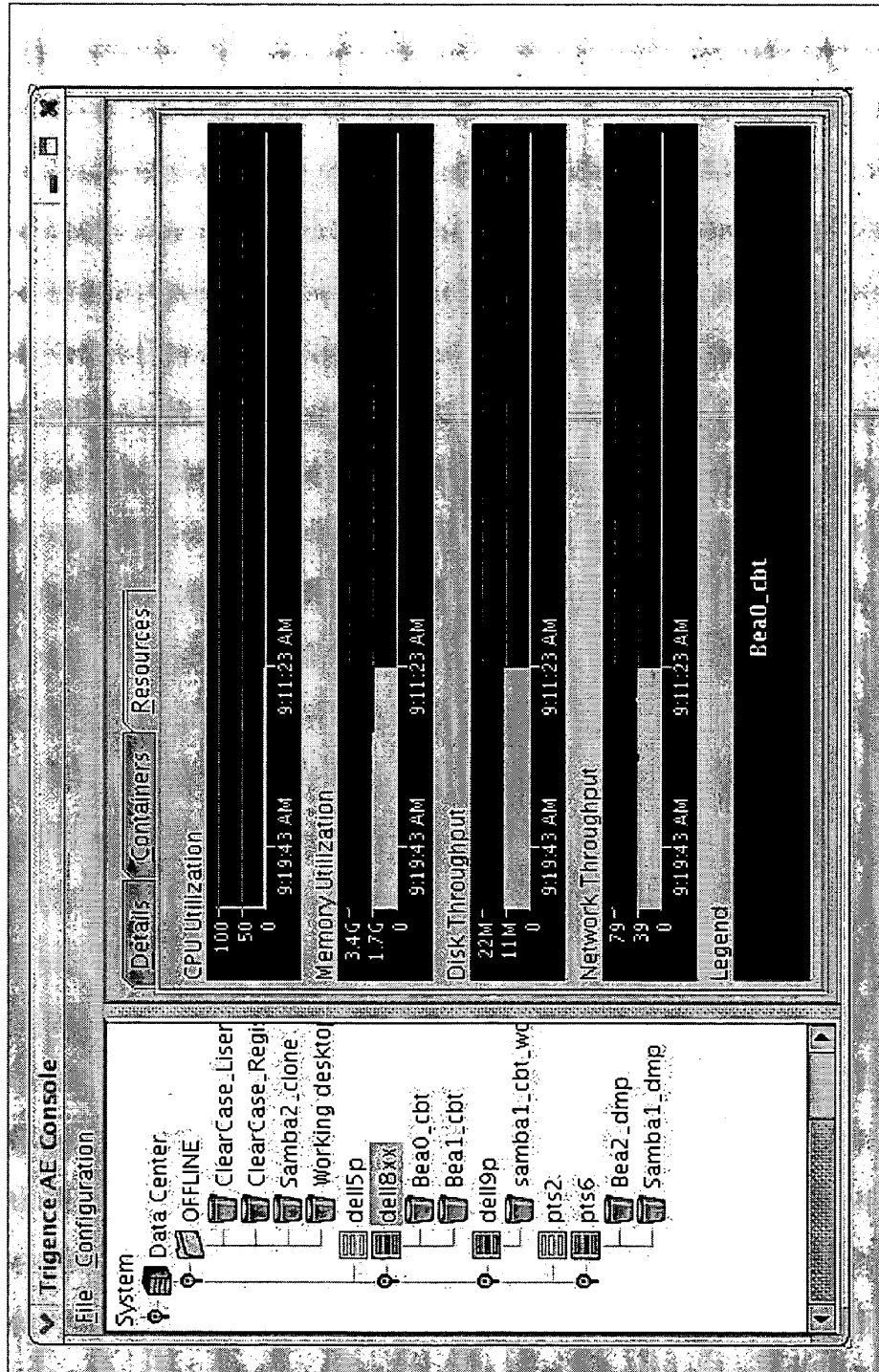


Figure 14

U.S. Patent

Apr. 14, 2009

Sheet 15 of 17

US 7,519,814 B2

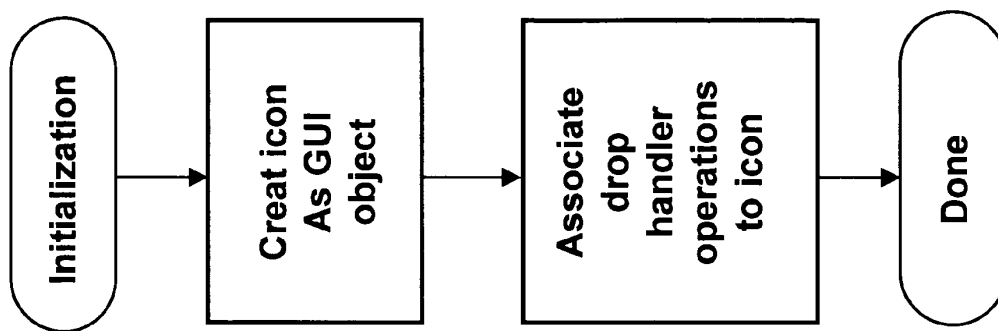


Figure 15

U.S. Patent

Apr. 14, 2009

Sheet 16 of 17

US 7,519,814 B2

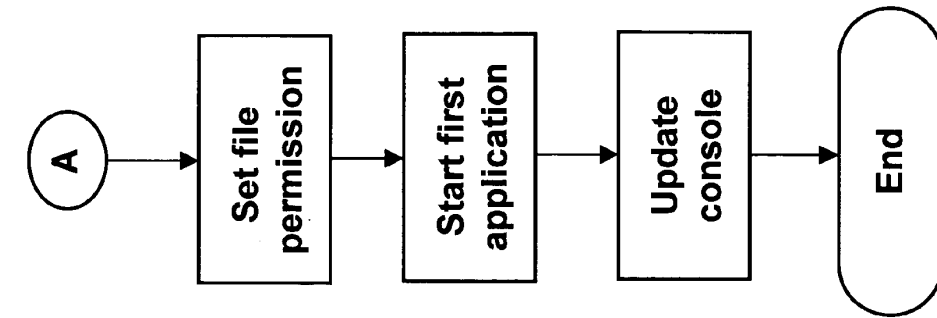
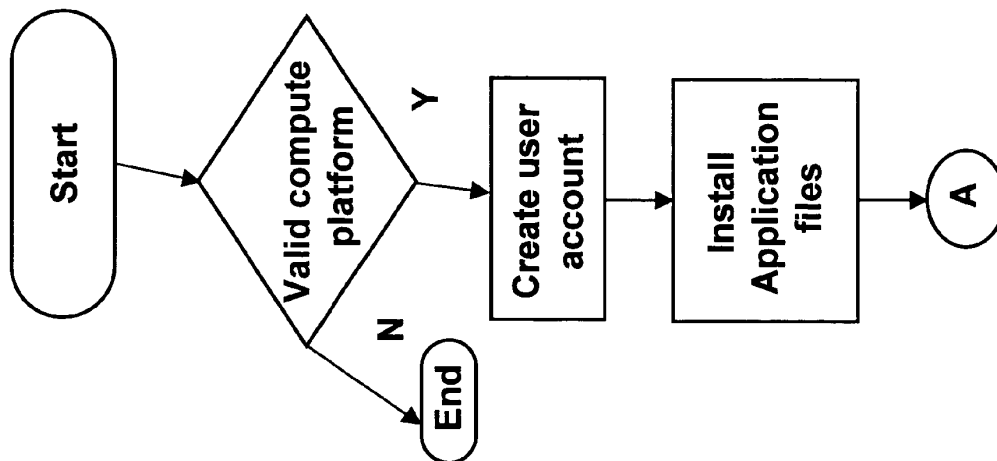


Figure 16



U.S. Patent

Apr. 14, 2009

Sheet 17 of 17

US 7,519,814 B2

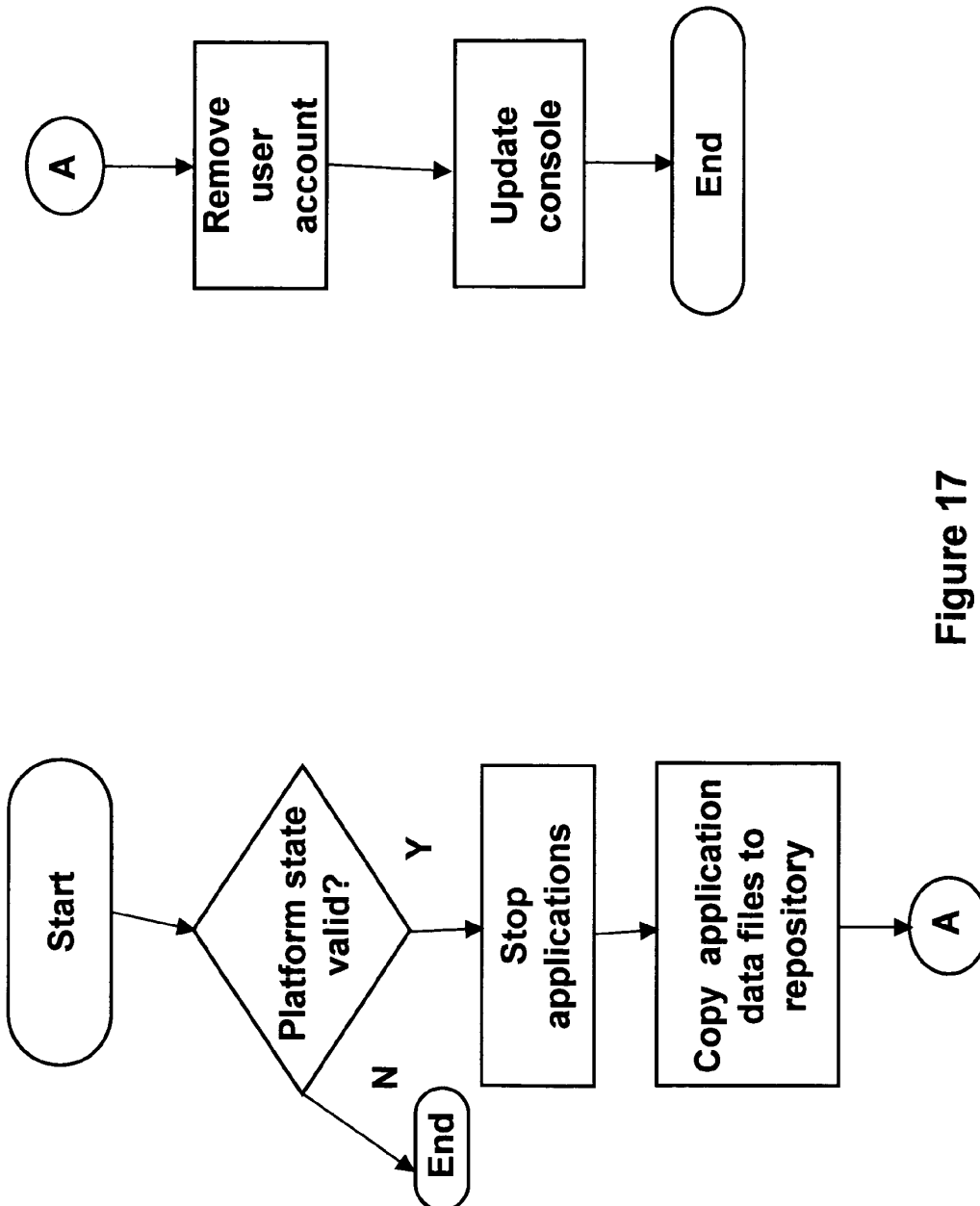


Figure 17

US 7,519,814 B2

1

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**CROSS-REFERENCE TO RELATED APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

2

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

```
/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1
```

US 7,519,814 B2

3

/lib/libnsl.so.1
 /lib/libdl.so.2
 /lib/i686/libpthread.so.0
 /lib/i686/libc.so.6
 /lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
 /etc/httpd/conf
 /etc/httpd/conf.d
 /etc/httpd/logs
 /etc/httpd/modules
 /etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

4

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySQL and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

US 7,519,814 B2

5

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of

6

this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed “spoofing” and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. 3 is a pictorial diagram illustrating the creation of a container.

FIG. 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the “kernel module” which is used to handle system calls.

FIG. 5 is a diagram similar to FIG. 4, wherein additional configuration data is provided.

FIG. 6 is a diagram illustrating a system wherein the containers are secure containers.

FIG. 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. 9 is a diagram that illustrates the installation of a container on a server.

FIG. 10 is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. 14 is a screen snapshot of an implementation according to the schematic of FIG. 13.

FIG. 15 is a flow chart of a method of initializing icons of FIG. 14.

FIG. 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. 14; and,

FIG. 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13.

DETAILED DESCRIPTION

Turning now to FIG. 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c.

US 7,519,814 B2

7

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. 2 illustrates a system in accordance with the invention in which multiple applications **21a**, **21b**, **21c**, **21d**, **21e**, and **21f** can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20a**, **20b** and **20c**. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20a**, **20b** and **20c** are shown in FIG. 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. 4 illustrates a system having two secure containers **20a** **20b**, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20a**, **20b** environment system calls are intercepted, by the kernel module **43** passing through services installed as part of

8

this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

As introduced heretofore, “spoofing” is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system’s kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the ‘uname’ system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has ‘spoofed’ the ‘uname’ system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same “spoofing” mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

- tracking applications, for example a tracking of which applications are in and out of a specific container;
- spoofing particular values returned by the kernel;
- applying resource checks and imposing resource limits;
- monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,
- retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20a** or **20b**. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. 6 that each application executing associated with a container **20a** or **20b**, or contained within a

US 7,519,814 B2

9

container **20a** or **20b**, is able to access files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20a** association are not able to access the files contained in another container **20b**.

When a container **20a** or **20b** is installed specific applications are started, as is shown in FIG. 7 and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. 7, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. 8, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. 9 shows that the files **82a**, **84a** are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing

10

the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. 9.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform and then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. 3, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

US 7,519,814 B2

11

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

- modifying the contents a file to add a container specific IP address;
- modifying the contents of a directory to define start scripts that should be executed;
- modifying the contents of a file to define which mount points will relate to a container;
- removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications

12

are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

US 7,519,814 B2

13

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

14

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

US 7,519,814 B2

15

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. 14, a screen snapshot of an implementation is shown according to the schematic of FIG. 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. 15, shown is a flow chart of a method of initializing the icons of FIG. 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. 16 are performed by an install drop handler. During installation, verification is performed to

16

determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. 15), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

US 7,519,814 B2

17

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

18

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

- a) running an instance of a service on a server;
- b) determining which files are being used; and,
- c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim 14 comprising the steps of:

- assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
- installing the container on a server; and,
- testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

- using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with a unique icon representing

US 7,519,814 B2

19

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim 20 further comprising starting a distributed software application.

22. A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim 17 further comprising creating a user account for the service.

25. A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim 1, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;
displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

20

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim 31, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

Exhibit 2

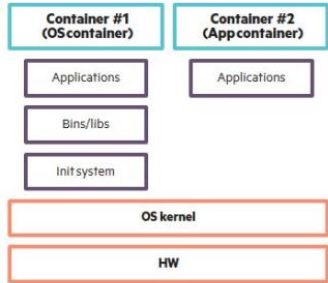
U.S. Patent No. 7,519,814 vs. HPE

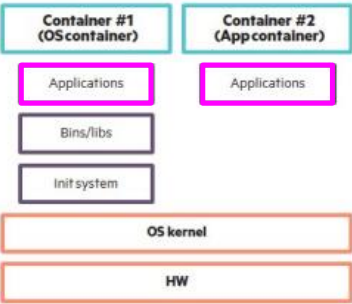
Accused Instrumentalities: HPE's Ezmeral Runtime Enterprise, and all versions and variations thereof since the issuance of the asserted patent.

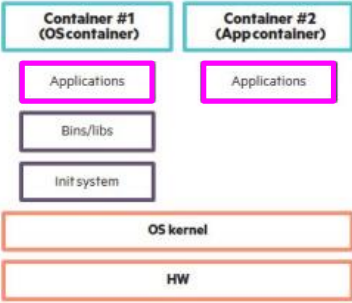
Claim 1

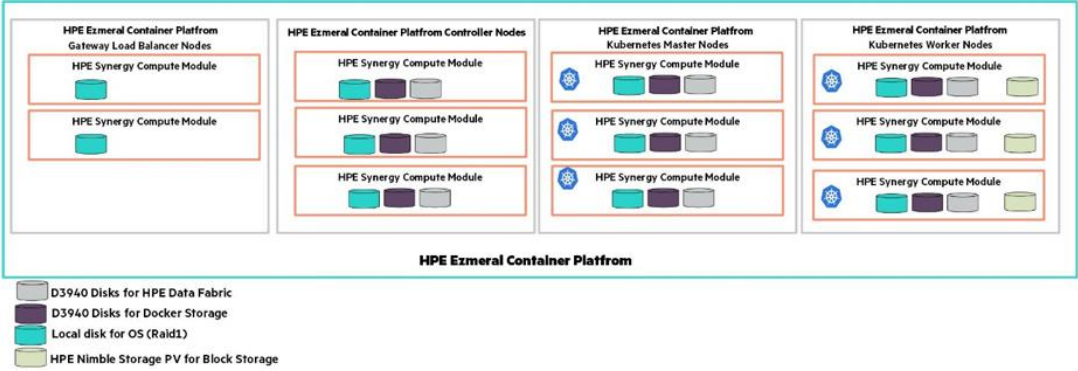
Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>To the extent the preamble is limiting, HPE practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>HPE Ezmeral Runtime Enterprise is an enterprise-grade container orchestration platform that is designed for the containerization of both cloud-native and non-cloud-native monolithic applications with persistent data. It deploys 100% open-source Kubernetes for orchestration, provides a state-of-the-art file system and data fabric for persistent container storage, and provides enterprises with the ability to deploy non-cloud-native AI and Analytics workloads in containers. Enterprises can now easily extend the agility and efficiency benefits of containers to more of their enterprise applications—running on either bare-metal or virtualized infrastructure, on-premises, in multiple clouds, or at the edge.</p> <p>https://www.hpe.com/psnow/doc/a50004264enw.pdf?jumpid=in_pdp-psnow-qs</p> <p>The offering formerly known as the HPE Ezmeral Container Platform is really focused on a lot more than just containers, and it provides businesses with more than just container orchestration software. The name change to HPE Ezmeral Runtime Enterprise reflects the fact that this is not just a solution for container platform orchestration. This platform offers an incredible wealth of capabilities and features you can use to modernize, deploy, monitor, and manage your applications.</p> <p>https://community.hpe.com/t5/hpe-ezmeral-uncut/hpe-ezmeral-container-platform-is-now-hpe-ezmeral-runtime/ba-p/7151720</p>

Claim 1	Accused Instrumentalities																
	<p>▪ OS agnostic – With an application and all its necessary files bundled into one unit – minus an operating system – the container can run on different operating systems, hardware, networks, storage systems and security policies. This means that any environment is compatible, so developers don't need to re-write applications for different servers.</p> <p>https://www.hpe.com/us/en/what-is/caas.html</p> <p>With HPE Ezmeral Runtime Enterprise running on HPE Synergy or HPE ProLiant servers, enterprises can extend the agility and efficiency benefits of containers to more of their enterprise applications—running on either bare metal or virtualized infrastructure, either on-premises, in multiple public clouds, or at the edge.</p> <p>https://www.hpe.com/psnow/doc/a50003599enw</p> <p>HPE Ezmeral Runtime Enterprise and HPE Ezmeral ML Ops Capabilities Matrix</p> <table><tr><th></th><th>HPE Ezmeral Runtime Enterprise Essentials</th><th>HPE Ezmeral Runtime Enterprise</th><th>HPE Ezmeral MLOps</th></tr><tr><td>Operating Systems (OS)</td><td>Yes</td><td>Yes</td><td>Yes</td></tr><tr><td>RHEL OS</td><td>Yes</td><td>Yes</td><td>Yes</td></tr><tr><td>SLES OS</td><td>Yes</td><td>Yes</td><td>Yes</td></tr></table> <p>https://www.hpe.com/psnow/doc/a50004264enw.pdf?jumpid=in_pdp-psnow-qs</p> <p>Standard Features</p> <ul style="list-style-type: none">• Leverages portability of containers to run on any infrastructure (HPE or non-HPE) and any public cloud <p>https://www.hpe.com/psnow/doc/a50004264enw.pdf?jumpid=in_pdp-psnow-qs</p>		HPE Ezmeral Runtime Enterprise Essentials	HPE Ezmeral Runtime Enterprise	HPE Ezmeral MLOps	Operating Systems (OS)	Yes	Yes	Yes	RHEL OS	Yes	Yes	Yes	SLES OS	Yes	Yes	Yes
	HPE Ezmeral Runtime Enterprise Essentials	HPE Ezmeral Runtime Enterprise	HPE Ezmeral MLOps														
Operating Systems (OS)	Yes	Yes	Yes														
RHEL OS	Yes	Yes	Yes														
SLES OS	Yes	Yes	Yes														

Claim 1	Accused Instrumentalities
	<p data-bbox="716 256 1031 277">Two Linux containers on a single system</p>  <p data-bbox="709 610 1923 675">https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf</p> <p data-bbox="709 711 1923 898">Each license allows the customer to deploy the HPE Ezmeral Container Platform on one Core and 2 terabytes of Storage Capacity. The customer must purchase more licenses if they exceed the allowable amount of Cores or Storage Capacity. As used in this Agreement, Core means a part of a CPU that executes a single stream of compiled instruction code. Each physical processor contains smaller processing units called physical CPU cores. Some processors have two cores, some four, some eight, and so on. Core capacity represents the total number of cores available within a given system. The number of cores is counted as the number of logical cores presented to the product guest OS. For licensing purposes, the number of cores on a given Ezmeral Container Platform host is the number of unique cores available to the kernel in the OS on which the Ezmeral Container Platform software is directly installed, regardless of the number of threads in each core. It equals the product of Core(s) per socket and Socket(s), as shown in the output of</p> <p data-bbox="709 906 1766 971">https://docs.ezmeral.hpe.com/runtime-enterprise/56/home/about-hpe-ezmeral-container-pl/GEN_End_User_Software_Agreement.html</p> <p data-bbox="716 1027 1856 1206">Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p data-bbox="709 1222 1457 1252">https://www.techtarget.com/searchdatacenter/definition/kernel</p> <p data-bbox="709 1292 1923 1417">Instead of using a hypervisor to manage VMs, the figure shows how containers isolate applications into separate environments (containers) that include processor, memory, and networking resources as part of the container itself. This environment provides OS-level virtualization. Containers have their own root; and, users and processes do not perform operations outside of the container environment. The host OS kernel manages container workloads directly, which reduces the overhead involved with managing system resources. This improves efficiency and therefore, improves performance.</p>

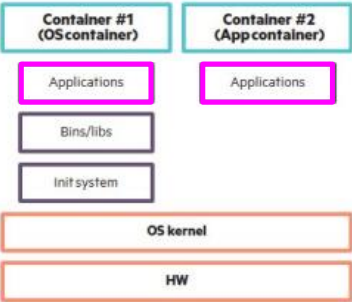
Claim 1	Accused Instrumentalities
	<p data-bbox="743 256 1062 277">Two Linux containers on a single system</p>  <p data-bbox="714 623 1923 688">https://h50146.www5.hp.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf</p> <p data-bbox="718 727 1606 862">Docker container: A <i>Docker container</i> is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable <i>microservices</i>. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.</p> <p data-bbox="714 873 1923 932">https://support.hp.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html</p>
[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;	<p data-bbox="714 943 1892 1110">The method practiced by HPE through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p> <p data-bbox="714 1133 821 1166"><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Pod: For Kubernetes, a <i>pod</i> is a group of containers deployed on a single host.</p> <p>Data Fabric cluster: This is a Kubernetes cluster that is used for HPE Ezmeral Data Fabric storage. A Data Fabric cluster is a Custom Resource in Kubernetes that is supported by operators in HPE Ezmeral Runtime Enterprise.</p> <p>Data Fabric CR: This typically refers to the Custom Resource specification for a Data Fabric cluster that is supported by an HPE Ezmeral Runtime Enterprise <code>dataplat</code> operator. It specifies each type of pod that the cluster would comprise. The per-pod specification may include CPU, memory, disk, and port requirements. Together with node labels and annotations, the Data Fabric CR influences the placement and scheduling of cluster pods by Kubernetes. HPE Ezmeral Runtime Enterprise creates and applies the Data Fabric CR when creating the first Data Fabric cluster. The Data Fabric CR may be subsequently patched/modified when expanding the cluster, or by a user with suitable privileges.</p> <p>Docker container: A <i>Docker container</i> is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable <i>microservices</i>. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html</p> <p>Two Linux containers on a single system</p>  <p>https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf</p>

Claim 1	Accused Instrumentalities
	 <p>Storage</p> <p>The HPE Synergy D3940 Storage Module provides solid state disks and hard disk drives to local systems where it is consumed by HPE Ezmeral Data Fabric, and optionally by compute nodes as boot devices. HPE Nimble Storage dynamically provides Persistent Volume (PV) for containers using Dynamic Volume Provisioner which is integrated with the HPE CSI Driver.</p> <p>https://www.hpe.com/psnow/doc/a50002075enw</p> <p>HPE Nimble Storage</p> <p>HPE Nimble Storage AF40 is used to provide persistent, block storage in this solution. The HPE Nimble Storage array for Docker data provides the storage volume to host the repository, to store container images, and also provides persistent volume for applications.</p> <p>https://www.hpe.com/psnow/doc/a50002075enw</p> <p>Container images</p> <p>A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p>https://kubernetes.io/docs/concepts/containers/</p>

Claim 1	Accused Instrumentalities
	<p>An application container is a stand-alone, all-in-one package for a software application. Containers include the application binaries, plus the software dependencies and the hardware requirements needed to run, all wrapped up into an independent, self-contained unit.</p> <p>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/</p> <p>Because each application container creates an isolated environment for its application, the resources allocated to it are the entire machine. Other copies of the same container are "unaware" of each other.</p> <p>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/</p> <p>▪ OS agnostic – With an application and all its necessary files bundled into one unit – minus an operating system – the container can run on different operating systems, hardware, networks, storage systems and security policies. This means that any environment is compatible, so developers don't need to re-write applications for different servers.</p> <p>https://www.hpe.com/us/en/what-is/caas.html</p> <p>6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p>Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.</p> <p>https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p> <p>Kubernetes namespaces have the following uses:</p> <ul style="list-style-type: none"> • Isolation: Teams, projects, and customers exist in their own environment within a cluster, and do not impact each other's work. <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp55hen_us&docLocale=en_US&page=reference/universal-concepts/Namespaces.html</p>

Claim 1	Accused Instrumentalities
	<p>Using containers isolates software and allows it to work independently across different operating systems, hardware, networks, storage systems, and security policies. It allows the container-based application to transition seamlessly through development, testing, and production environments. Because an operating system is not packed into the container, each container uses minimal computing resources, making it light and easy to install.</p> <p>https://www.hpe.com/us/en/what-is/containers.html</p>
<p>[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,</p>	<p>In the method practiced by HPE through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.</p> <p><i>See, e.g.:</i></p> <p>Docker container: A <i>Docker container</i> is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable <i>microservices</i>. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html</p> <p>▪ OS agnostic – With an application and all its necessary files bundled into one unit – minus an operating system – the container can run on different operating systems, hardware, networks, storage systems and security policies. This means that any environment is compatible, so developers don't need to re-write applications for different servers.</p> <p>https://www.hpe.com/us/en/what-is/caas.html</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="743 256 1062 277">Two Linux containers on a single system</p>  <p data-bbox="714 623 1923 686">https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf</p>
<p data-bbox="184 716 579 776">[1c] the containers of application software excluding a kernel,</p>	<p data-bbox="714 716 1913 776">In the method practiced by HPE through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p data-bbox="714 802 821 829"><i>See, e.g.:</i></p> <p data-bbox="722 860 1923 1047">Docker container: A <i>Docker container</i> is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable <i>microservices</i>. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.</p> <p data-bbox="714 1065 1923 1125">https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html</p> <p data-bbox="722 1157 1440 1179">6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p data-bbox="722 1214 1682 1292">Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.</p> <p data-bbox="714 1308 1808 1336">https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Containers and VMs perform somewhat similar functions in that they provide virtualized environments in which software applications can run separately from the rest of the system. But these technologies are very different and are used in different situations. Each virtual machine runs both an OS and the application, while containers share a single OS via a kernel, making them more lightweight and portable.</p> <p>https://www.hpe.com/us/en/what-is/containers.html</p>
<p>[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by HPE through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.</p> <p><i>See, e.g.:</i></p> <p>Docker container: A <i>Docker container</i> is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable <i>microservices</i>. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html</p> <p>An application container is a stand-alone, all-in-one package for a software application. Containers include the application binaries, plus the software dependencies and the hardware requirements needed to run, all wrapped up into an independent, self-contained unit.</p> <p>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/</p> <ul style="list-style-type: none"> ▪ OS agnostic – With an application and all its necessary files bundled into one unit – minus an operating system – the container can run on different operating systems, hardware, networks, storage systems and security policies. This means that any environment is compatible, so developers don't need to re-write applications for different servers. <p>https://www.hpe.com/us/en/what-is/caas.html</p>
<p>[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by HPE through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Docker container: A <i>Docker container</i> is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable <i>microservices</i>. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.</p> <p>https://support.hpe.com/hpsc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html</p> <p><code>COPY</code> and <code>ADD</code> : These commands copy files and directories from your local filesystem into the Docker image. They are often used to include your application code, configuration files, and dependencies.</p> <p>https://medium.com/@swalperen3008/what-is-dockerize-and-dockerize-your-project-a-step-by-step-guide-899c48a34df6</p> <h3>Container images</h3> <p>A <i>container image</i> is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p>https://kubernetes.io/docs/concepts/containers/</p>
[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,	<p>In the method practiced by HPE through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p><i>See, e.g.:</i></p> <p>Docker container: A <i>Docker container</i> is a lightweight, standalone, executable software package that runs specific services. This software package includes code, runtime, system libraries, configurations, etc. that run as an isolated process in user space. A Docker container is typically used to deploy scalable and repeatable <i>microservices</i>. HPE Ezmeral Runtime Enterprise contains innovations around storage, networking, and security to utilize Docker containers as lightweight virtual machines to run Big Data and analytics applications.</p>

Claim 1	Accused Instrumentalities
	<p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html</p> <p>Kubernetes namespaces have the following uses:</p> <ul style="list-style-type: none"> • Isolation: Teams, projects, and customers exist in their own environment within a cluster, and do not impact each other's work. <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp55hen_us&docLocale=en_US&page=reference/universal-concepts/Namespace.html</p> <p>Because each application container creates an isolated environment for its application, the resources allocated to it are the entire machine. Other copies of the same container are "unaware" of each other.</p> <p>https://developer.hpe.com/blog/kubernetes-application-containers-managing-containers-and-cluster-resour/</p>
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by HPE through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p><i>See, e.g.:</i></p> <p>Using containers isolates software and allows it to work independently across different operating systems, hardware, networks, storage systems, and security policies. It allows the container-based application to transition seamlessly through development, testing, and production environments. Because an operating system is not packed into the container, each container uses minimal computing resources, making it light and easy to install.</p> <p>https://www.hpe.com/us/en/what-is/containers.html</p> <p>Node storage: <i>Node storage</i> is storage space available for backing the root file systems of containers. Each HPE Ezmeral Runtime Enterprise host contributes node storage space that is used by the virtual nodes (Docker containers) assigned to that host. The Platform Administrator may optionally specify a quota limiting how much node storage a tenant's virtual nodes may consume.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_Definitions.html</p>

Exhibit 3



US007784058B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,784,058 B2**
(45) **Date of Patent:** **Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**

2002/0004854 A1 1/2002 Hartley
2002/0174215 A1 11/2002 Schaefer 709/224
2003/0101292 A1 5/2003 Fisher
2004/0025165 A1* 2/2004 Desoli et al. 719/310

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002
WO WO 2006/039181 A 4/2006

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

OTHER PUBLICATIONS

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1293 days.

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

* cited by examiner

(21) Appl. No.: **10/946,536**

Primary Examiner—Hyung S Sough

Assistant Examiner—Syed Roni

(22) Filed: **Sep. 21, 2004**

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A. Attorneys at Law

(65) **Prior Publication Data**

US 2005/0066303 A1 Mar. 24, 2005

(57) **ABSTRACT**

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep. 22, 2003.

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

(51) **Int. Cl.**
G06F 9/22 (2006.01)

(52) **U.S. Cl.** **719/310**; 719/319

(58) **Field of Classification Search** 719/310,
719/319

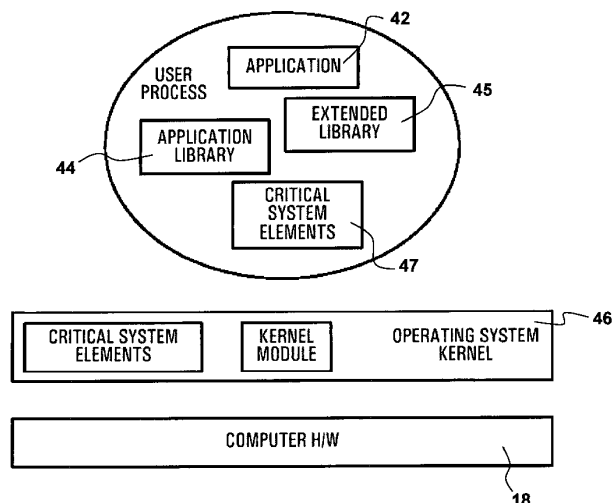
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,481,706 A * 1/1996 Peek 710/200
6,212,574 B1 * 4/2001 O'Rourke et al. 719/321
6,260,075 B1 * 7/2001 Cabrero et al. 719/310

18 Claims, 7 Drawing Sheets



U.S. Patent

Aug. 24, 2010

Sheet 1 of 7

US 7,784,058 B2

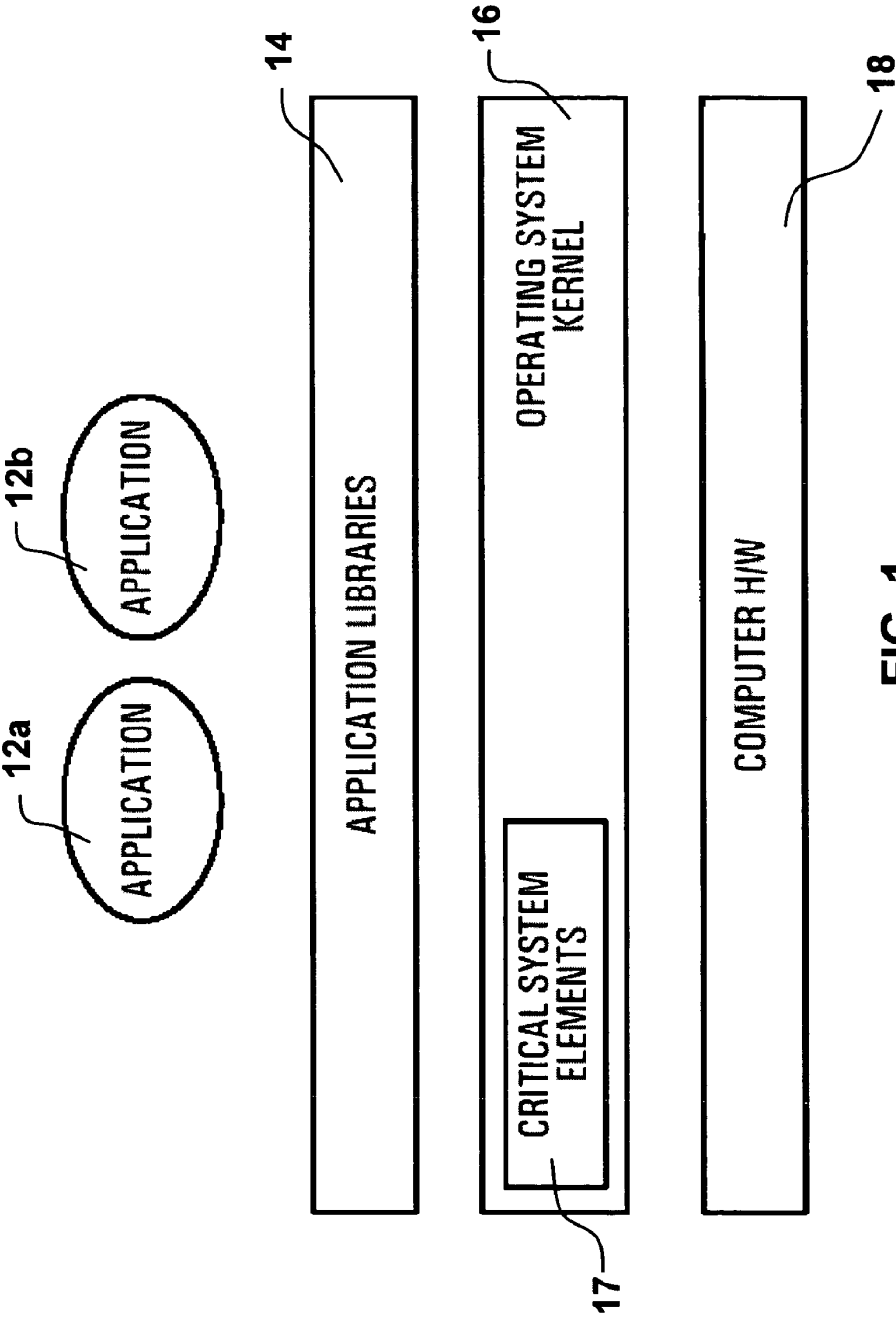


FIG. 1
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 2 of 7

US 7,784,058 B2

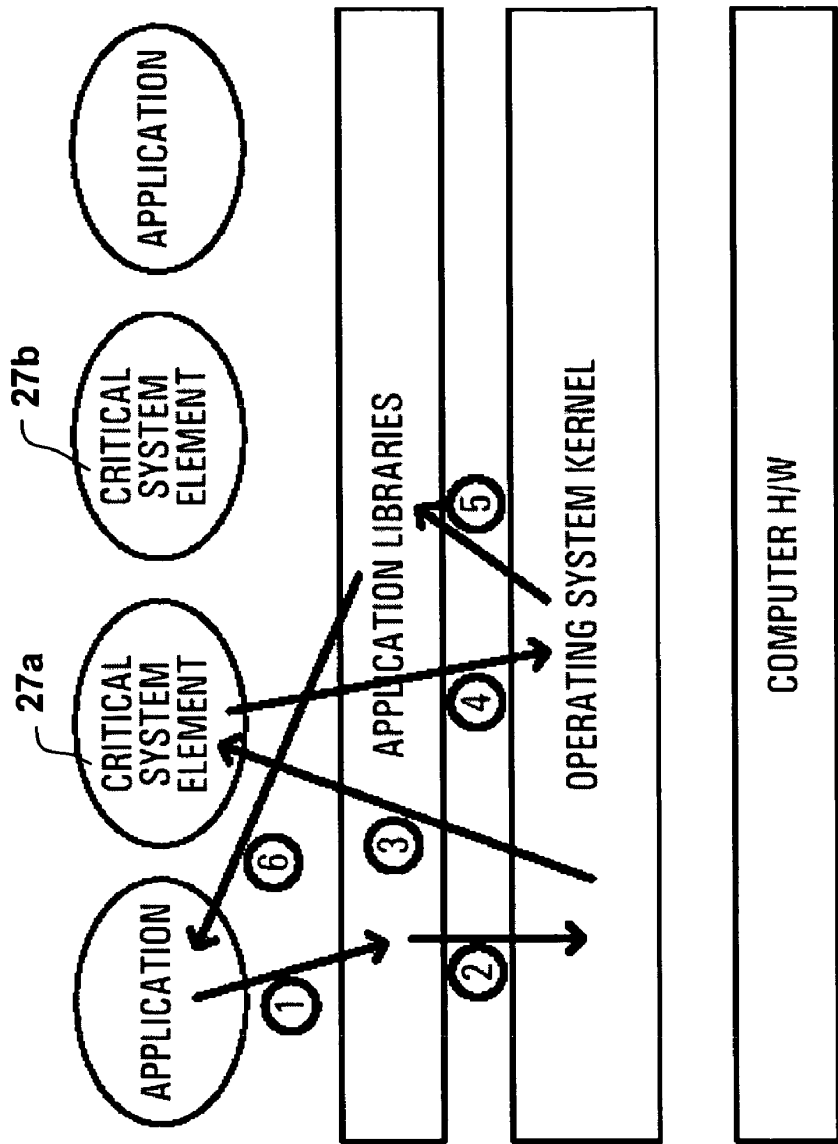


FIG. 2a
Prior Art

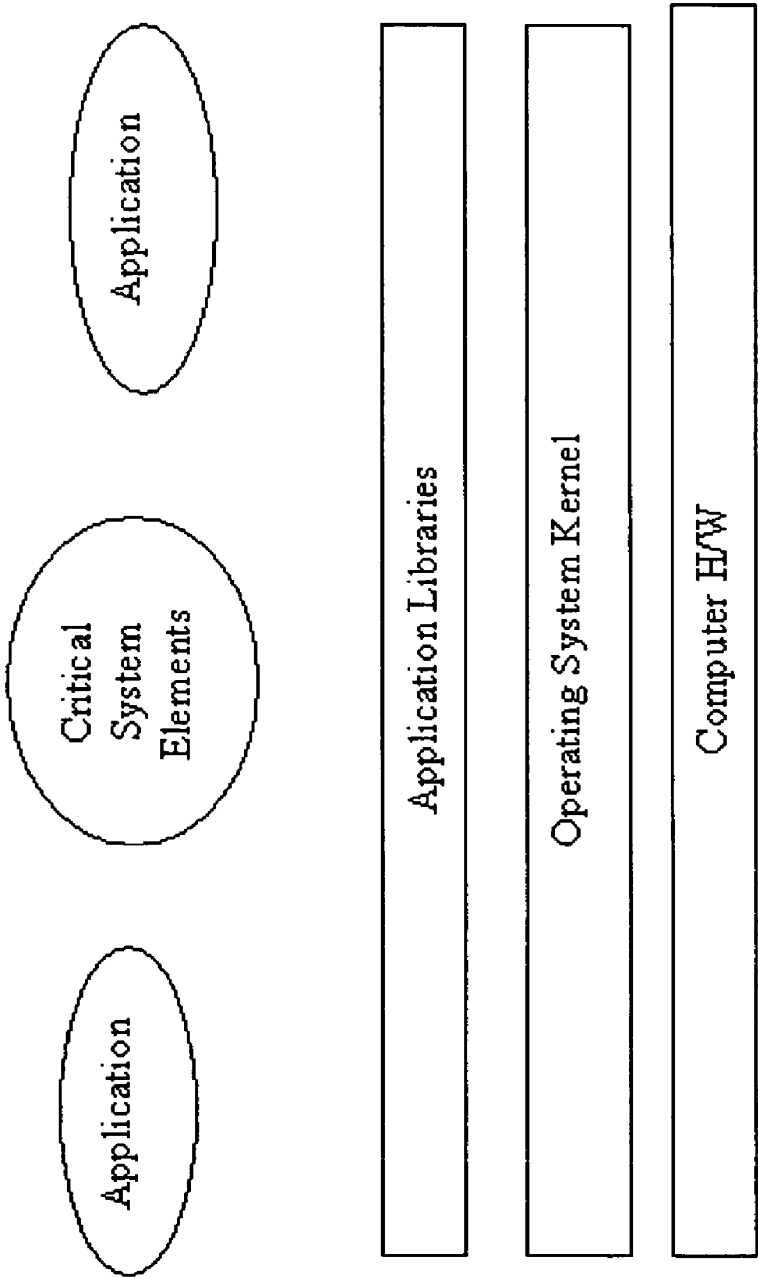


FIG. 2b
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 4 of 7

US 7,784,058 B2

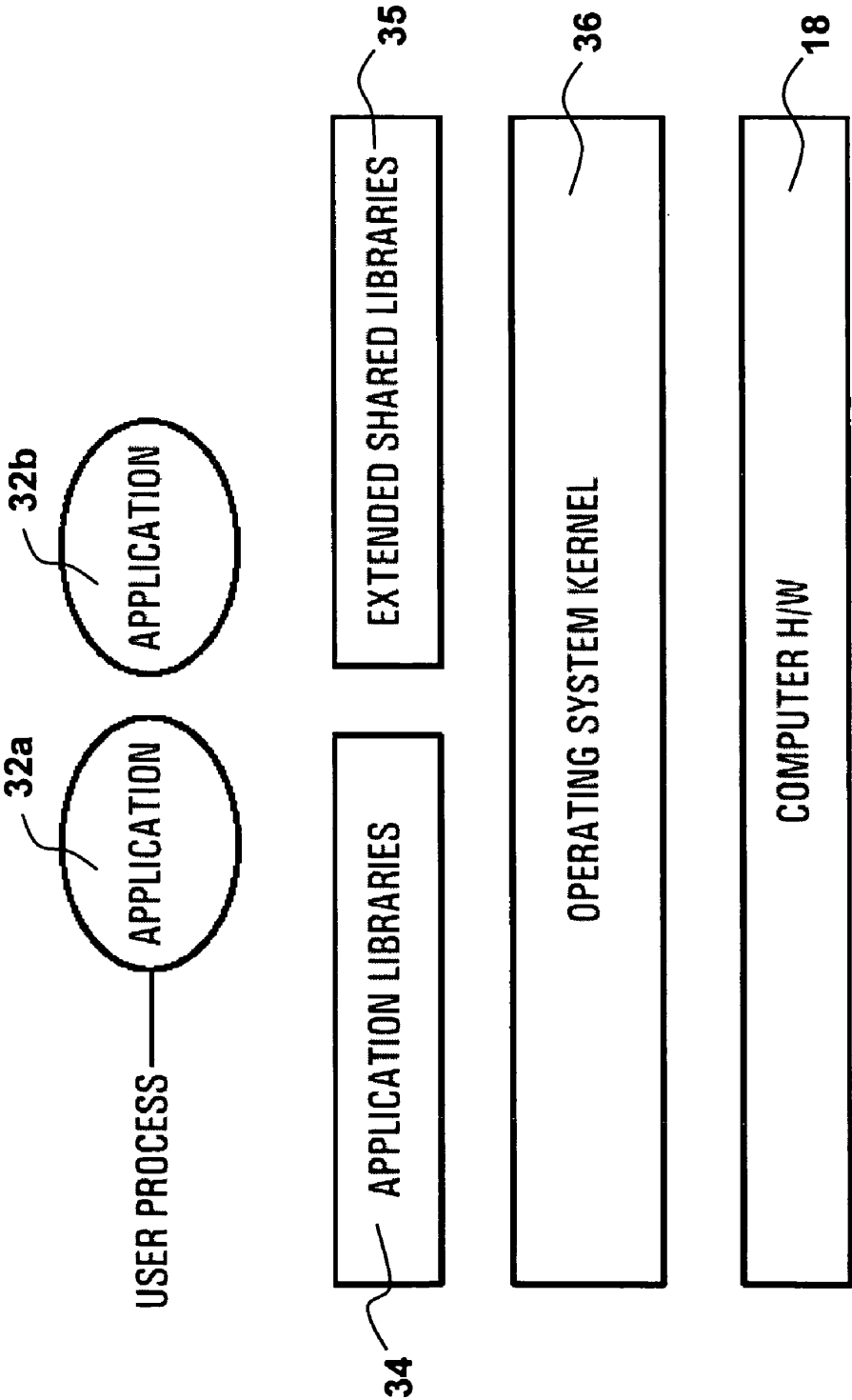


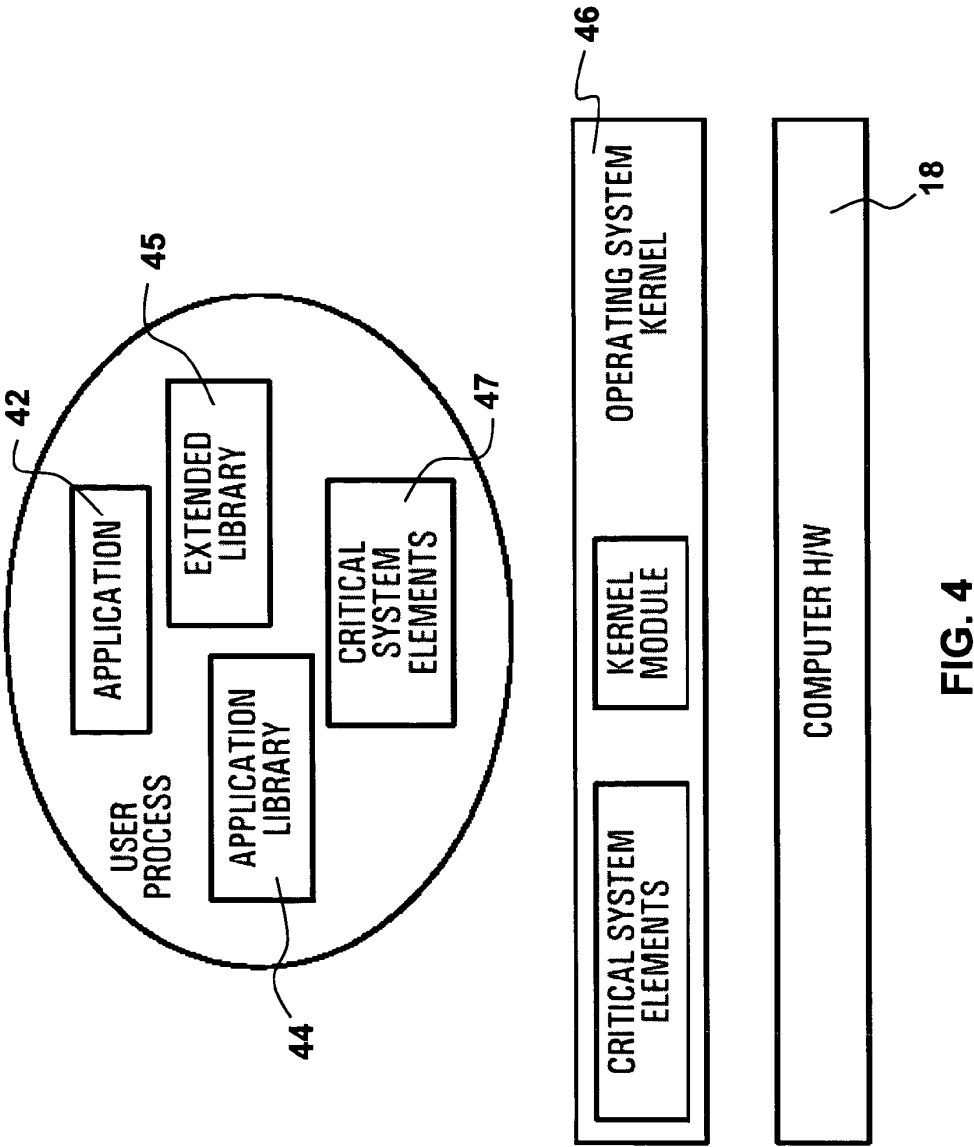
FIG. 3

U.S. Patent

Aug. 24, 2010

Sheet 5 of 7

US 7,784,058 B2



U.S. Patent

Aug. 24, 2010

Sheet 6 of 7

US 7,784,058 B2

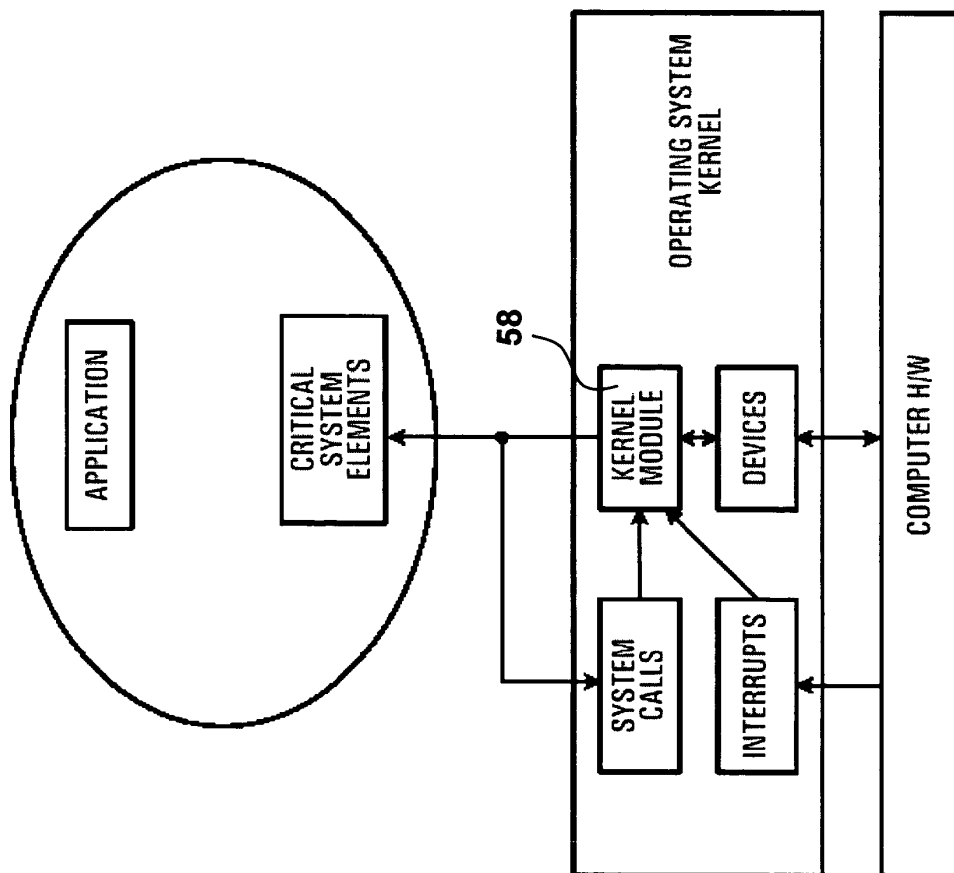


FIG. 5

U.S. Patent

Aug. 24, 2010

Sheet 7 of 7

US 7,784,058 B2

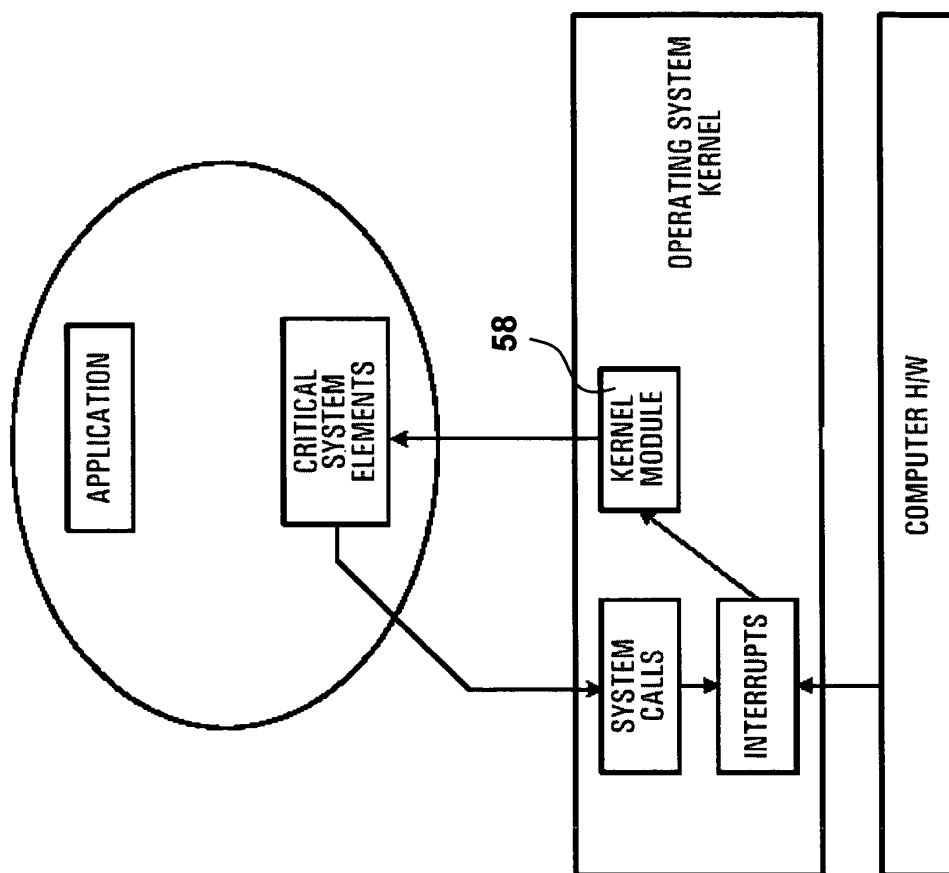


FIG. 6

US 7,784,058 B2

1

**COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled “User Mode Critical System Element as Shared Libs”, which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

2

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.

2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

US 7,784,058 B2

3

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

4

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. 1 is an architectural view of the traditional monolithic prior art operating system;

US 7,784,058 B2

5

FIG. 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. 3 is an architectural view of an embodiment of the invention;

FIG. 4 is a functional view showing how critical system elements exist in the same context as an application;

FIG. 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. 6 shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TCP/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP

6

stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality

2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided

2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b

US 7,784,058 B2

7

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12a**, **12b** makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12a**, **12b** in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2a** shows a system architecture where critical system elements **27a**, **27b** execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2a** is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2a** as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2b** is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2a** and FIG. **2b** is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2b** is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the

8

case with a multi-server architecture depicted in FIG. **2a**. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32a**, **32b**, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45,47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

US 7,784,058 B2

9

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. 4. FIG. 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

10

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:

- a) a processor;
- b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,
- c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

US 7,784,058 B2

11

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

6. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

7. A computing system according to claim 6 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

8. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

12

9. A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

10. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

11. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

12. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

13. A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

14. A computing system according to claim 13 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

15. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

16. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

17. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.

* * * * *

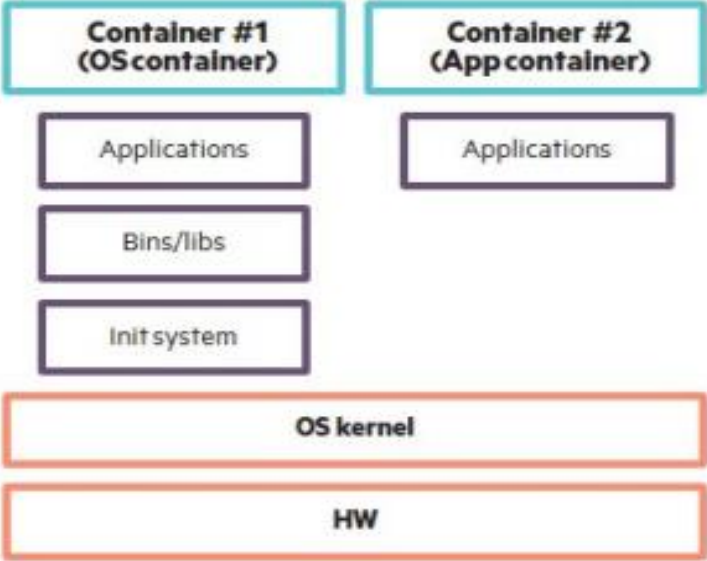
Exhibit 4

U.S. Patent No. 7,784,058 vs. HPE

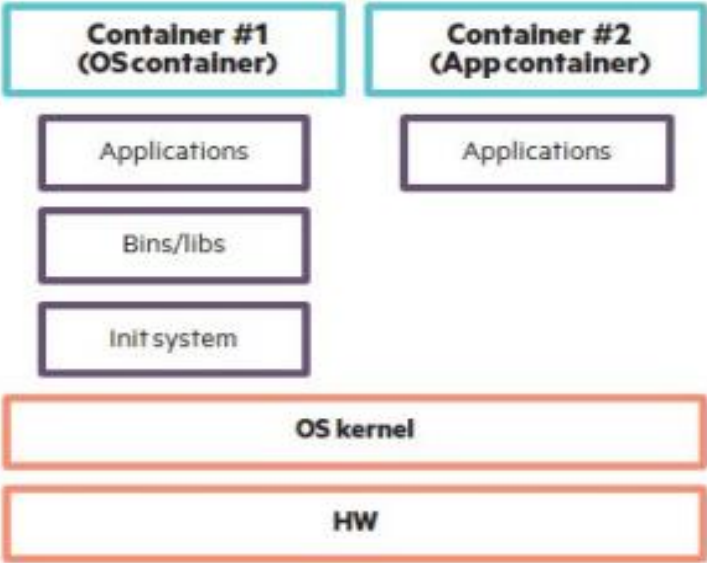
Accused Instrumentalities: HPE's Ezmeral Runtime Enterprise, and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
<p>[1pre] 1. A computing system for executing a plurality of software applications comprising:</p>	<p>To the extent the preamble is limiting, each Accused Instrumentality comprises or constitutes a computing system for executing a plurality of software applications as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>HPE Ezmeral Runtime Enterprise is a unified platform built on open-source Kubernetes and designed for both cloud-native applications and non-cloud-native applications running on any infrastructure; whether on-premises, in multiple public clouds, in a hybrid model, or at the edge.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&page=home/about-hpe-ezmeral-container-pl/Welcome.html</p> <p>Containers provide the core runtime abstraction for the user applications. These containers provide isolation between user applications and the rest of the infrastructure. The containers are based on Docker.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00097165en_us&docLocale=en_US&page=GUID-6B6676DB-AF5F-4555-B6AB-D2C11A89F320.html</p>

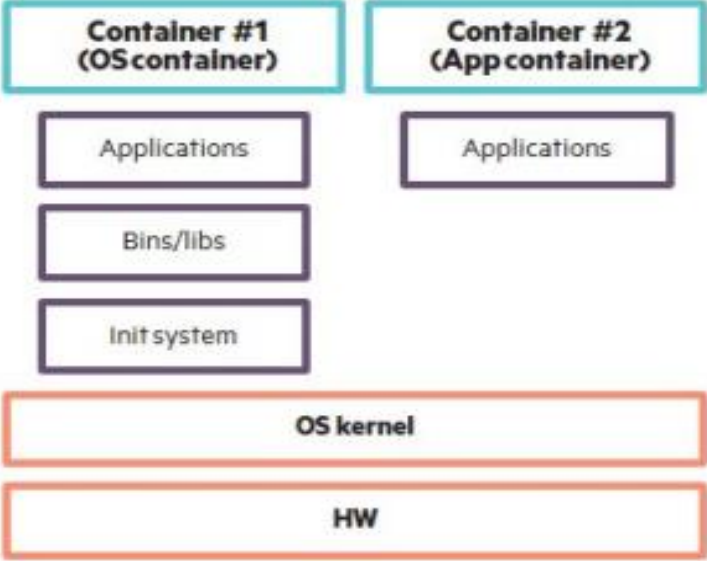
Claim 1	Accused Instrumentalities
	<p style="text-align: center;">Two Linux containers on a single system</p>  <p>https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf</p>
[1a] a) a processor;	<p>Each Accused Instrumentality comprises a processor.</p> <p><i>See, e.g.:</i></p> <p>Each license allows the customer to deploy the HPE Ezmeral Container Platform on one Core and 2 terabytes of Storage Capacity. The customer must purchase more licenses if they exceed the allowable amount of Cores or Storage Capacity. As used in this Agreement, Core means a part of a CPU that executes a single stream of compiled instruction code. Each physical processor contains smaller processing units called physical CPU cores. Some processors have two cores, some</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_End_User_Software_Agreement.html</p>

Claim 1	Accused Instrumentalities								
[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,	<p>Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.</p> <p>See, e.g.:</p> <p>Containers provide the core runtime abstraction for the user applications. These containers provide isolation between user applications and the rest of the infrastructure. The containers are based on Docker.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00097165en_us&docLocale=en_US&page=GUID-6B6676DB-AF5F-4555-B6AB-D2C11A89F320.html</p> <p>Each license allows the customer to deploy the HPE Ezmeral Container Platform on one Core and 2 terabytes of Storage Capacity. The customer must purchase more licenses if they exceed the allowable amount of Cores or Storage Capacity. As used in this Agreement, Core means a part of a CPU that executes a single stream of compiled instruction code. Each physical processor contains smaller processing units called physical CPU cores. Some processors have two cores, some</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&docLocale=en_US&page=home/about-hpe-ezmeral-container-pl/GEN_End_User_Software_Agreement.html</p> <p>HPE Ezmeral Runtime Enterprise supports the following operating systems:</p> <table><tr><th>HPE Ezmeral Runtime Enterprise Version</th><th>CentOS Support</th><th>RHEL Support</th><th>SUSE Support</th></tr><tr><td></td><td></td><td></td><td></td></tr></table> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00ecp54hen_us&page=home/about-hpe-ezmeral-container-pl/GEN_OS_Support.html</p>	HPE Ezmeral Runtime Enterprise Version	CentOS Support	RHEL Support	SUSE Support				
HPE Ezmeral Runtime Enterprise Version	CentOS Support	RHEL Support	SUSE Support						

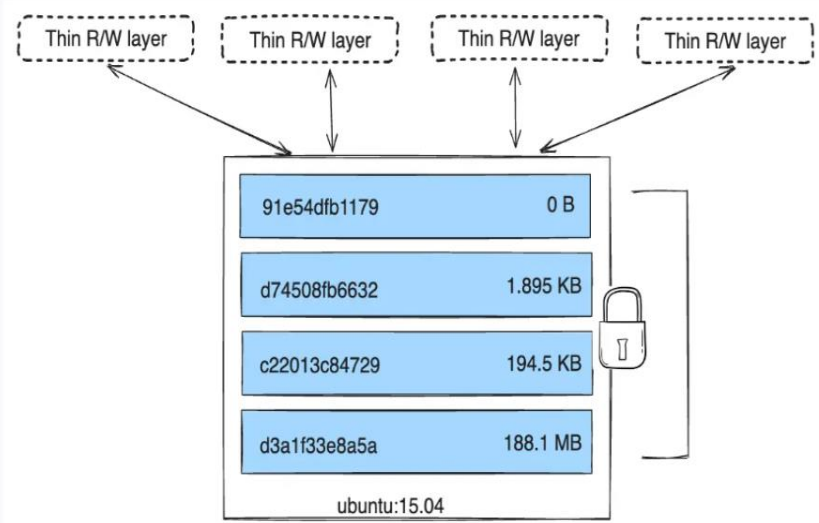
Claim 1	Accused Instrumentalities
	<p data-bbox="758 269 1398 302">Two Linux containers on a single system</p>  <p data-bbox="711 943 1923 1008">https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf</p> <p data-bbox="730 1049 905 1081">Kernel mode</p> <p data-bbox="730 1110 1911 1292">Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p data-bbox="711 1312 1457 1344">https://www.techtarget.com/searchdatacenter/definition/kernel</p>

Claim 1	Accused Instrumentalities
<p>[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and</p>	<p>Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.</p> <p><i>See, e.g.:</i></p> <p>Containers provide the core runtime abstraction for the user applications. These containers provide isolation between user applications and the rest of the infrastructure. The containers are based on Docker.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00097165en_us&docLocale=en_US&page=GUID-6B6676DB-AF5F-4555-B6AB-D2C11A89F320.html</p> <p>The container starts with a base image, and the microservice is packaged into a container image and then deployed through the container platform. The container platform is based on</p> <p>https://www.hpe.com/us/en/what-is/container-platform.html</p> <p>Container images</p> <p>A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p>https://kubernetes.io/docs/concepts/containers/</p> <p>The idea of containerization is to isolate and package the application with all the dependencies in a container.</p> <p>https://community.hpe.com/t5/hpe-blog-uk-ireland-middle-east/containerization-the-next-generation-of-virtualization/ba-p/7154442</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="716 256 1562 427"><u>Container image files</u> are complete, static and executable versions of an application or service and differ from one technology to another. <u>Docker images</u> are made up of multiple layers, which start with a base image that includes all of the dependencies needed to execute code in a container. Each image has a readable/writable layer on top of static unchanging layers. Because each container has its own specific container layer that customizes that specific container, underlying image layers can be saved and reused in multiple containers. An Open Container Initiative (OCI)</p> <p data-bbox="716 440 1881 500">https://www.techtarget.com/searchitoperations/definition/container-containerization-or-container-based-virtualization</p> <p data-bbox="716 537 1898 656">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p> <div data-bbox="919 719 1740 1239"></div> <p data-bbox="716 1265 1268 1291">https://docs.docker.com/storage/storagedriver/</p>

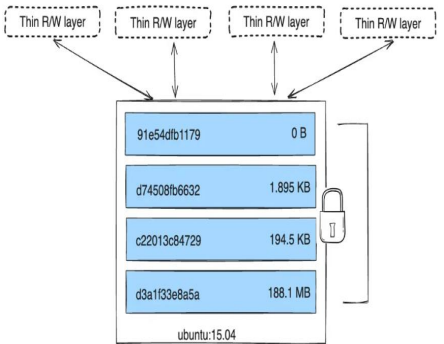
Claim 1	Accused Instrumentalities
	<p style="text-align: center;">Two Linux containers on a single system</p>  <p style="text-align: center;"> Container #1 (OS container) Container #2 (App container) </p> <p style="text-align: center;"> Applications Applications </p> <p style="text-align: center;"> Bins/libs </p> <p style="text-align: center;"> Init system </p> <p style="text-align: center;"> OS kernel </p> <p style="text-align: center;"> HW </p> <p>https://h50146.www5.hp.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf</p>
<p>[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,</p>	<p>In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.</p> <p>For example, a Docker base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and dependencies, including system libraries or shared library critical system elements (SLCSEs). When the container runs the image, it creates a runtime instance of that container image.</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="716 256 821 289"><i>See, e.g.:</i></p> <p data-bbox="716 321 1913 410">Hewlett Packard Enterprise provides publicly available base OS images for use in containerized clusters. These images extend the base OS images available from Docker hub by adding several packages that permit HPE Ezmeral Runtime Enterprise to manage container orchestration seamlessly and to improve the security of the container.</p> <p data-bbox="716 414 1740 479">https://docs.ezmeral.hpe.com/runtime-enterprise/55/app-workbench-5-1/custom-base-images/AWB51_About_Custom_Base_Images.html</p> <p data-bbox="716 527 1892 560">The idea of containerization is to isolate and package the application with all the dependencies in a container.</p> <p data-bbox="716 576 1923 641">https://community.hpe.com/t5/hpe-blog-uk-ireland-middle-east/containerization-the-next-generation-of-virtualization/ba-p/7154442</p> <h2 data-bbox="751 678 1115 732">Container images</h2> <p data-bbox="751 764 1446 902">A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p data-bbox="716 930 1272 963">https://kubernetes.io/docs/concepts/containers/</p> <p data-bbox="716 1011 1709 1166">Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p data-bbox="716 1190 1558 1222">https://www.techtarget.com/searchitoperations/definition/Docker-image</p>

Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p> <p>Containers only have access to resources that are defined in the image, https://www.hpe.com/us/en/what-is/docker.html</p>
<p>[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications</p>	<p>In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.</p> <p>When a Docker image is used to create a container, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each</p>

Claim 1	Accused Instrumentalities
<p>running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and</p>	<p>container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications running in their respective containers. The docker image includes essential system files, libraries, and dependencies required to run the software application within the container. The Docker containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it operates in isolation and runs its own instance of the software application without sharing resources or critical system elements with other containers. This ensures that each container has its own isolated context. Docker containers can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same or a different function.</p> <p><i>See, e.g.:</i></p> <p>Containers provide the core runtime abstraction for the user applications. These containers provide isolation between user applications and the rest of the infrastructure. The containers are based on Docker.</p> <p>https://support.hpe.com/hpesc/public/docDisplay?docId=a00097165en_us&docLocale=en_US&page=GUID-6B6676DB-AF5F-4555-B6AB-D2C11A89F320.html</p>

Claim 1	Accused Instrumentalities
	<p>Two Linux containers on a single system</p> <p>https://h50146.www5.hpe.com/products/software/oe/linux/mainstream/support/whitepaper/pdfs/4AA6-2761ENW.pdf</p>

Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p>
<p>[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p>	<p>In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p> <p>For example, In Docker, each container operates independently, and a Docker base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When a Docker image is used to create a container in ECS, an instance of</p>

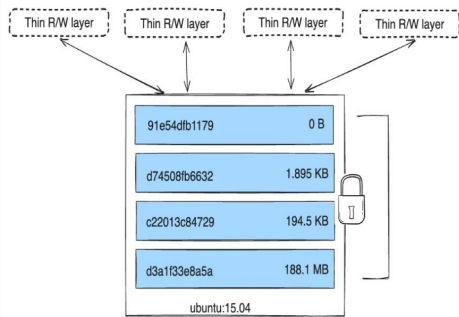
Claim 1	Accused Instrumentalities
	<p>the SLCSE is provided to a software application. Therefore, different instances of the SLCSE are provided to different applications for performing either a same or a different function, simultaneously.</p> <p><i>See, e.g.:</i></p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image, Last accessed on June 14, 2023</p> <p>A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.</p> <p>https://docs.docker.com/get-started/overview/</p> <p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>

EXHIBIT C

**UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION**

VIRTAMOVE, CORP.,

Plaintiff,

v.

GOOGLE LLC,

Defendant.

Case No. 7:24-cv-00033-DC-DTG

JURY TRIAL DEMANDED

**FIRST AMENDED COMPLAINT FOR PATENT INFRINGEMENT AGAINST
GOOGLE LLC**

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, “Plaintiff” or “VirtaMove”) makes the following allegations against Defendant Google LLC (“Defendant” or “Google”):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant’s unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058 (collectively, the “Asserted Patents”). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as Appzero Software Corp. (“Appzero”), which was established in

2010.

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repackage, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove's mission is to move and modernize the world's server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach \$2.1 billion in 2019 and \$4.3 billion in 2022—a compound annual growth rate (“CAGR”) of 30%. *See, e.g.*, <https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022>. Another source reported the application containers market had a market size of \$5.45 billion in 2024 and estimated it to reach \$19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.*, <https://www.mordorintelligence.com/industry-reports/application-container-market>.

6. Google LLC is a wholly-owned subsidiary of Alphabet, Inc, and a Delaware limited liability company with a principal place of business at 1600 Amphitheatre Parkway, Mountain View, California 94043. Google LLC may be served with process through its registered agent,

Becky DeGeorge, at 2710 Gateway Oaks Drive, Sacramento, California 95833. Google LLC is registered to do business in the State of Texas and has been since at least November 17, 2006.

JURISDICTION AND VENUE

7. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

9. Venue is proper in this District under 28 U.S.C. § 1400(b). Upon information and belief, Defendant has transacted business in this District and has committed acts of direct and indirect infringement in this District by, among other things, making, using, offering to sell, selling, and importing products that infringe the asserted patents. Defendant has at least one regular and established place of business in the District. For example, Google invested \$20 million to build a corporate office at 500 West 2nd Street, Austin, Texas 78701.

ADDITIONAL FACTS

10. Representatives of VirtaMove met with representatives of Google in 2015, 2020, and 2021 for the purposes of partnering with VirtaMove, demonstrating AppZero, training Google on how to use AppZero, allowing Google to run and evaluate AppZero, discussing integration of

AppZero into Google Cloud, and sharing materials about how AppZero works. During this time frame, Google would have learned that AppZero was patented as a matter of basic due diligence, or Google intentionally chose to remain willfully blind to this fact. VirtaMove's technology from its demos and disclosures appears to have made their way into the Accused Products of Google described below.

COUNT I

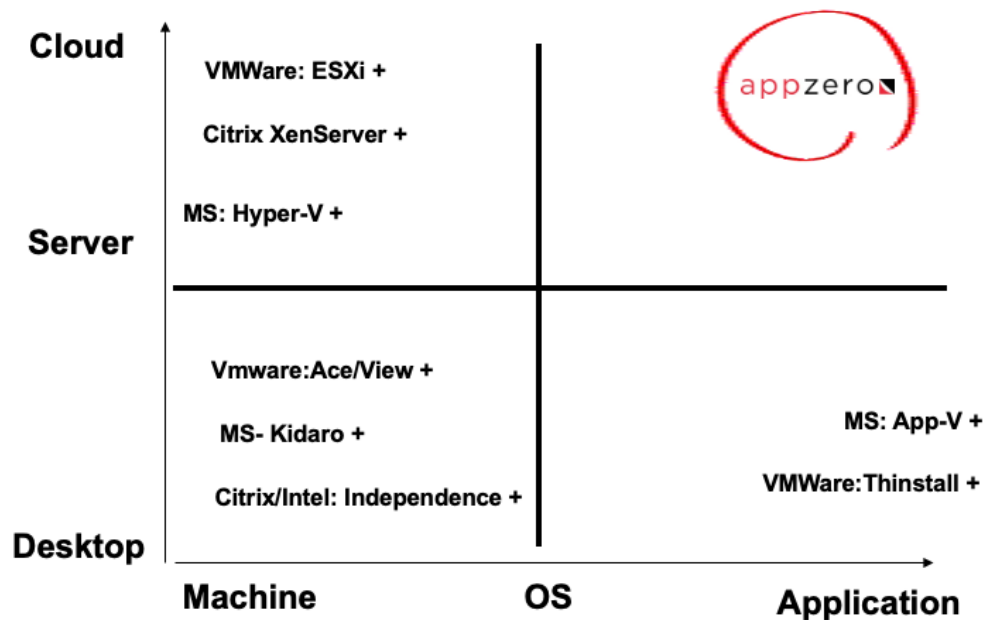
INFRINGEMENT OF U.S. PATENT NO. 7,519,814

11. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

12. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 ('814 patent), titled "System for Containerization of Application Sets," issued on April 14, 2009. A true and correct copy of the '814 Patent is attached as Exhibit 1.

13. The technology in the '814 Patent was not routine or conventional. Independent business intelligence provider Gartner reported that AppZero had no direct technological competitors in "Cool Vendors in Cloud Computing" in April 2009.

Virtualization Landscape



Note: No direct competition per Gartner "Cool Vendors in Cloud Computing" 4/09/2009

14. Additionally, the USPTO examined the patented technology and concluded that the prior art "fails to anticipate, disclose, teach, or suggest alone, or in combination, at the time of the invention, the features as set forth of the claim Nowhere in the prior art is found collectively the *italicized* claim elements (i.e., the various aspects of applications software not being sharable between the plurality of secure (and isolated) containers of application software, and unique root file systems different from an operating system's root file system, so as to allow for different versions of the same operating running on the same system/server environment), at the *time of the invention*, serving to patentably distinguish the prior art." Ex. 5 at 2–5. Thus, multiple aspects of the claims were not found anywhere in the prior art of record, alone or in combination, such that they cannot be described as "routine" or "conventional." *Id.*

15. The technology claimed is not directed to an abstract idea. Rather, it is directed to

a technological improvement in computer technology, and it provides a technological solution to a fundamentally technological problem. Indeed, the problem of application portability across operating systems is inherently rooted in computer technology with no non-technological analog, nor can the solution be performed solely within the human mind. In part, the “invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.” Ex. 1 at 2:13–15. This is despite how “a collection of applications . . . must be separated with each application installed on an individual computer system,” and despite how “certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.” *Id.* at 1:27–41. Moreover, the technology is distinguished from the Virtual Machine approach in that, with Virtual Machines, “an operating system, including files and a kernel, must be deployed for each application” while the technology of the ’814 patent requires only requires one operating system regardless of the number of application containers deployed, and it “offers the ability for applications to more effectively share a common compute platform, and also allow applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.” *Id.* at 1:51–2:3

16. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products (“Accused Products”), such as, e.g., Google’s Migrate to Containers, that directly infringe, literally and/or under the doctrine of equivalents, claims of the ’814 patent, for example:

Use Migrate to Containers to modernize traditional applications away from virtual machine (VM) instances and into native containers that run on Google Kubernetes Engine (GKE), GKE Enterprise clusters, or Cloud Run platform. You can migrate workloads from VMs that run on VMware or Compute Engine, giving you the flexibility to containerize your existing workloads with ease. Migrate to Containers supports modernization of IBM WebSphere, JBoss, Apache, Tomcat, WordPress, Windows IIS applications, as well as containerisation of Linux-based applications.

<https://cloud.google.com/migrate/containers/docs/getting-started>. Thus, Google directly infringes by its making, using (e.g., running on Google's servers), offering for sale (e.g., offering Migrate to Containers), selling (e.g., selling Migrate to Containers), and/or importing. Google makes, uses, offers for sale, sells, and/or imports Migrate to Containers, as described in <https://www.youtube.com/watch?v=Sbv-lzlGeIo>, <https://cloud.google.com/migrate/containers>, <https://cloud.google.com/migrate/containers/docs/getting-started>. Migrate to Containers is a tool to containerize existing VM-based applications to run on [Google Kubernetes Engine \(GKE\)](#), [GKE Autopilot clusters](#), [GKE Enterprise](#), or [Cloud Run](#). <https://cloud.google.com/migrate/containers/docs/anthos-migrate-benefits>.

17. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement of the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

18. Defendant also knowingly and intentionally induces infringement of claims of the '814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and/or earlier. Despite this knowledge of the '814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '814 patent. Defendant does so, knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '814 patent, thereby specifically intending for and inducing its customers to infringe the '814 patent through the customers' normal

and customary use of the Accused Products.

19. Defendant has also infringed, and continue to infringe, claims of the '814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '814 patent, in violation of 35 U.S.C. § 271(c).

20. The Accused Products satisfy all claim limitations of claims of the '814 patent. A claim chart comparing an independent claim of the '814 patent to a representative Accused Product, is attached as Exhibit 2, which is hereby incorporated by reference in its entirety.

21. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the '814 patent pursuant to 35 U.S.C. § 271.

22. As a result of Defendant's infringement of the '814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. See <https://web.archive.org/web/20230924012816/https://virtamove.com/about-us/product-patents>.

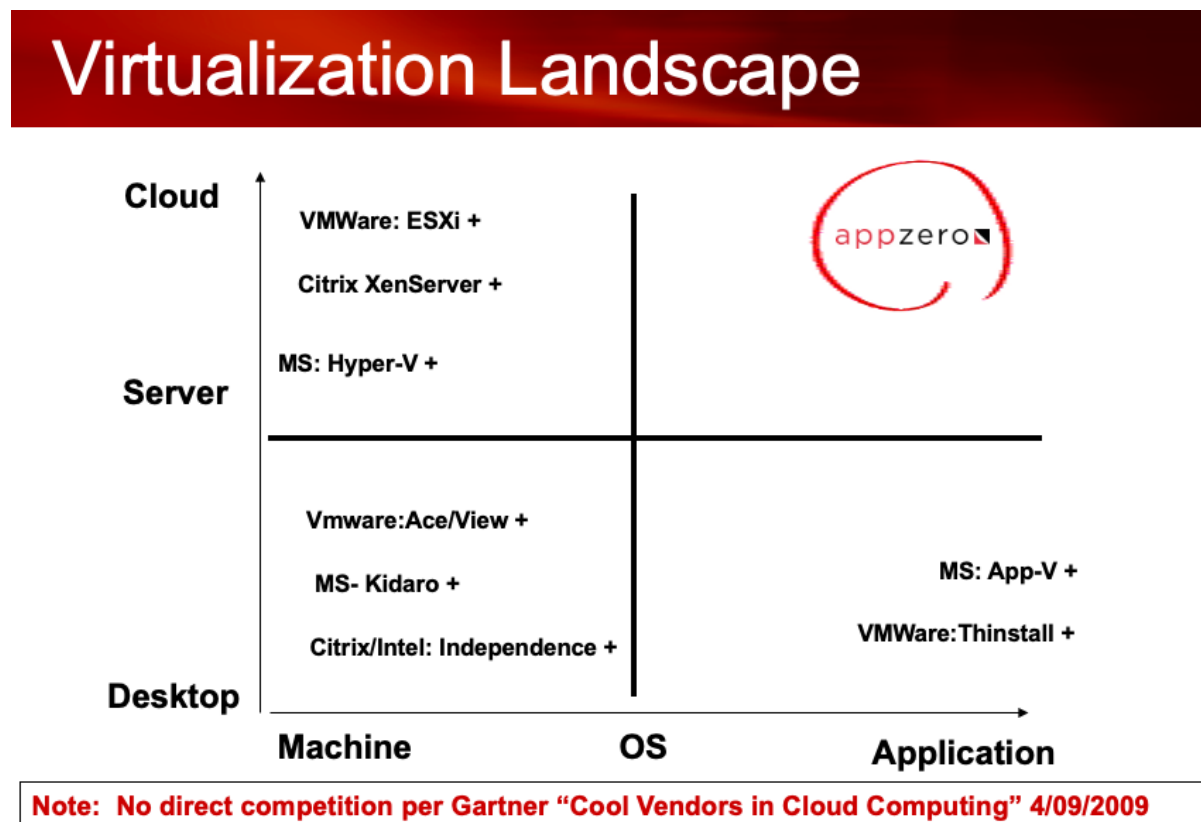
COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

23. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

24. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 ('058 patent), titled "Computing System Having User Mode Critical System Elements as Shared Libraries," issued on August 24, 2010. A true and correct copy of the '058 patent is attached as Exhibit 3.

25. The technology in the '058 Patent was not routine or conventional. Independent business intelligence provider Gartner reported that AppZero had no direct technological competitors in "Cool Vendors in Cloud Computing" in April 2009.



26. Additionally, the USPTO examined the patented technology and concluded that the none of the prior art discloses "a shared library having shared library critical system elements

(SLCSEs) stored therein for use by the plurality of software applications in user mode and i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, and ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing the same function. iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.” Ex. 6 at 4–5. Thus, multiple aspects of the claims were not found anywhere in the prior art of record, alone or in combination, such that they cannot be described as “routine” or “conventional.” *Id.*

27. The technology claimed is not directed to an abstract idea. Rather, it is directed to a technological improvement in computer technology, and it provides a technological solution to a fundamentally technological problem. Indeed, the problem of application portability across operating systems is inherently rooted in computer technology with no non-technological analog, nor can the solution be performed solely within the human mind. In part, the technology allows “a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE.” Ex. 3 at 1:46–50. This is despite how “two software

applications that require the same file, yet each requires a different version of the file will conflict,” and despite how “two applications that require independent access to specific network services will conflict.” *Id.* at 1:27–41. Moreover, the technology overcomes the “performance and operational differences” suffered by other technological solutions. *Id.* at 1:41–45.

28. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products (“Accused Products”), such as, e.g., Google’s Migrate to Containers, that directly infringe, literally and/or under the doctrine of equivalents, claims of the ’058 patent, for example:

Use Migrate to Containers to modernize traditional applications away from virtual machine (VM) instances and into native containers that run on Google Kubernetes Engine (GKE), GKE Enterprise clusters, or Cloud Run platform. You can migrate workloads from VMs that run on VMware or Compute Engine, giving you the flexibility to containerize your existing workloads with ease. Migrate to Containers supports modernization of IBM WebSphere, JBoss, Apache, Tomcat, WordPress, Windows IIS applications, as well as containerisation of Linux-based applications.

<https://cloud.google.com/migrate/containers/docs/getting-started>. Thus, Google directly infringes by its making, using (e.g., running on Google’s servers), offering for sale (e.g., offering Migrate to Containers), selling (e.g., selling Migrate to Containers), and/or importing. Google makes, uses, offers for sale, sells, and/or imports Migrate to Containers, as described in <https://www.youtube.com/watch?v=Sbv-lzlGeIo>, <https://cloud.google.com/migrate/containers>, <https://cloud.google.com/migrate/containers/docs/getting-started>. Migrate to Containers is a tool to containerize existing VM-based applications to run on [Google Kubernetes Engine \(GKE\)](#), [GKE Autopilot clusters](#), [GKE Enterprise](#), or [Cloud Run](#). <https://cloud.google.com/migrate/containers/docs/anthos-migrate-benefits>.

29. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement of the Asserted Patents, conditioning benefits on

participation in the infringement and establishing the timing and manner of the infringement.

30. Defendant also knowingly and intentionally induces infringement of claims of the '058 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed. Despite this knowledge of the '058 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '058 patent. Defendant does so, knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '058 patent, thereby specifically intending for and inducing its customers to infringe the '058 patent through the customers' normal and customary use of the Accused Products.

31. Defendant has also infringed, and continue to infringe, claims of the '058 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '058 patent, in violation of 35 U.S.C. § 271(c).

32. The Accused Products satisfy all claim limitations of claims of the '058 patent. A claim chart comparing an independent claim of the '058 patent to a representative Accused Product, is attached as Exhibit 4, which is hereby incorporated by reference in its entirety.

33. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the '058 patent pursuant to 35 U.S.C. § 271.

34. As a result of Defendant's infringement of the '058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. See <https://web.archive.org/web/20230924012816/https://virtamove.com/about-us/product-patents>.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

- a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;
- b. A permanent injunction prohibiting Defendant from further acts of infringement of each of the '814 and '058 patents;
- c. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant's infringement of each of the Asserted Patents;
- d. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;
- e. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys' fees against

Defendant; and

- f. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury of any issues so triable by right.

Dated: May 21, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)

rmirzaie@raklaw.com

Marc A. Fenster (CA SBN 181067)

mfenster@raklaw.com

Neil A. Rubin (CA SBN 250761)

nrubin@raklaw.com

Amy E. Hayden (CA SBN 287026)

ahayden@raklaw.com

Christian W. Conkle (CA SBN 306374)

cconkle@raklaw.com

Jacob Buczko (CA SBN 269408)

jbuczko@raklaw.com

Daniel Kolko (CA SBN 341680)

dkolko@raklaw.com

James Milkey (CA SBN 281283)

jmilkey@raklaw.com

James Tsuei (CA SBN 285530)

jtsuei@raklaw.com

Jonathan Ma (CA SBN 312773)

jma@raklaw.com

RUSS AUGUST & KABAT

12424 Wilshire Boulevard, 12th Floor

Los Angeles, CA 90025

Telephone: (310) 826-7474

Qi (Peter) Tong (TX SBN 24119042)

ptong@raklaw.com

RUSS AUGUST & KABAT

4925 Greenville Ave., Suite 200

Dallas, TX 75206

Telephone: (310) 826-7474

Attorneys for Plaintiff VirtaMove, Corp.

CERTIFICATE OF SERVICE

I certify that on May 21, 2024, a true and correct copy of the foregoing document was electronically filed with the Court and served on all parties of record via the Court's CM/ECF system.

/s/ Reza Mirzaie

Reza Mirzaie

Exhibit 1



US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,519,814 B2**
(45) **Date of Patent:** **Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

WO WO 2006/039181 A 4/2006

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248; 709/214; 719/319

(58) **Field of Classification Search** 713/167; 719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

OTHER PUBLICATIONS

Soltész, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667.*

Primary Examiner—Kambiz Zand

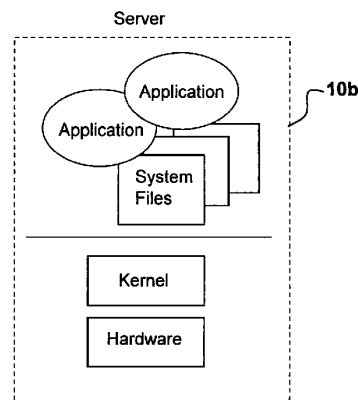
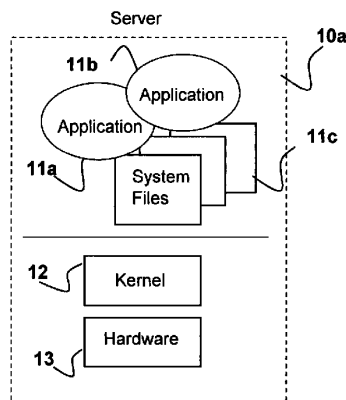
Assistant Examiner—Ronald Baum

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



US 7,519,814 B2

Page 2

U.S. PATENT DOCUMENTS				2002/0004854 A1	1/2002	Hartley	
6,847,970 B2 *	1/2005	Keller et al.	707/100	2002/0174215 A1	11/2002	Schaefer	709/224
7,076,784 B1 *	7/2006	Russell et al.	719/315	2003/0101292 A1	5/2003	Fisher	
7,287,259 B2 *	10/2007	Grier et al.	719/331	* cited by examiner			

U.S. Patent

Apr. 14, 2009

Sheet 1 of 17

US 7,519,814 B2

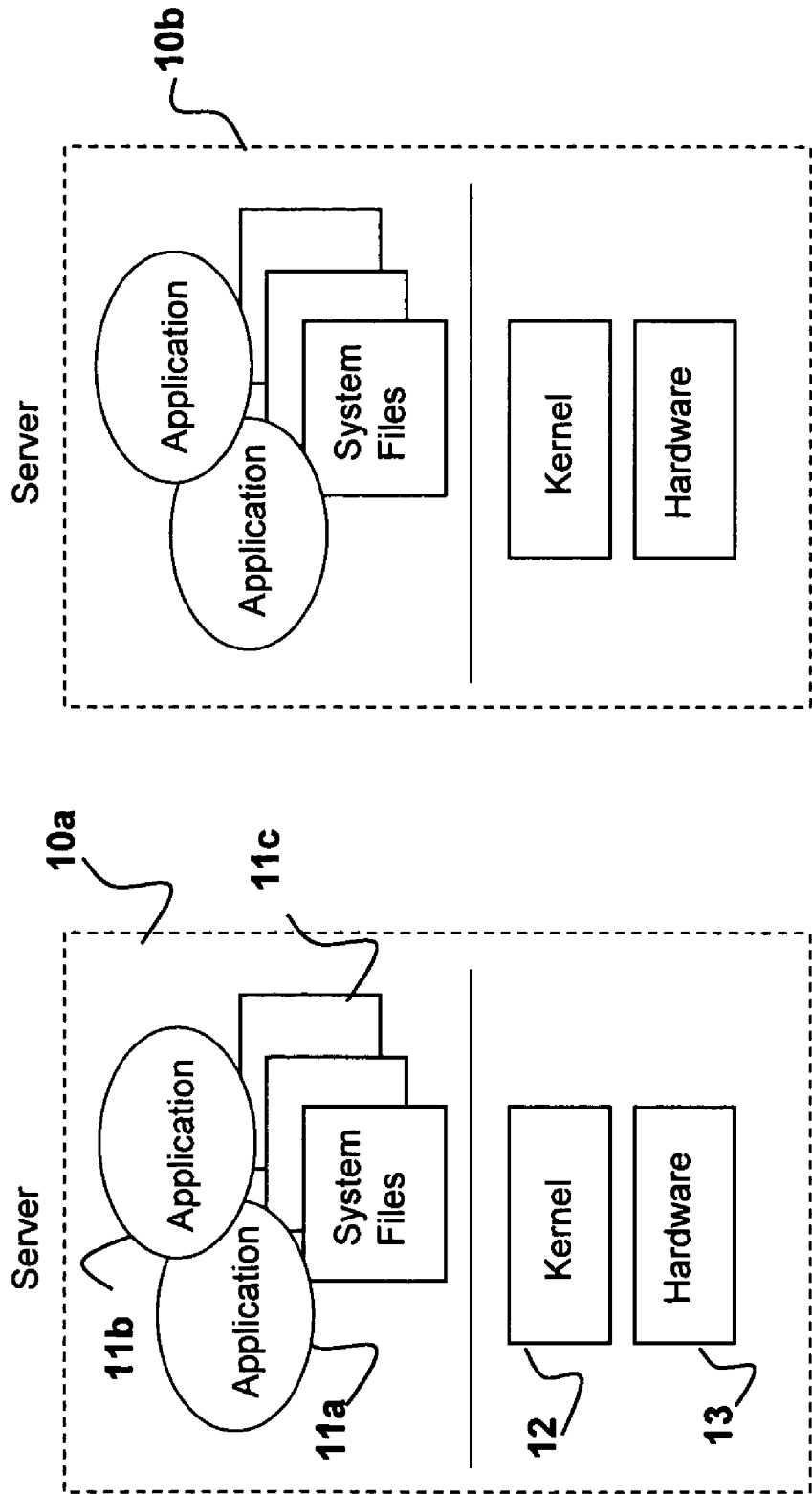


Figure 1

U.S. Patent

Apr. 14, 2009

Sheet 2 of 17

US 7,519,814 B2

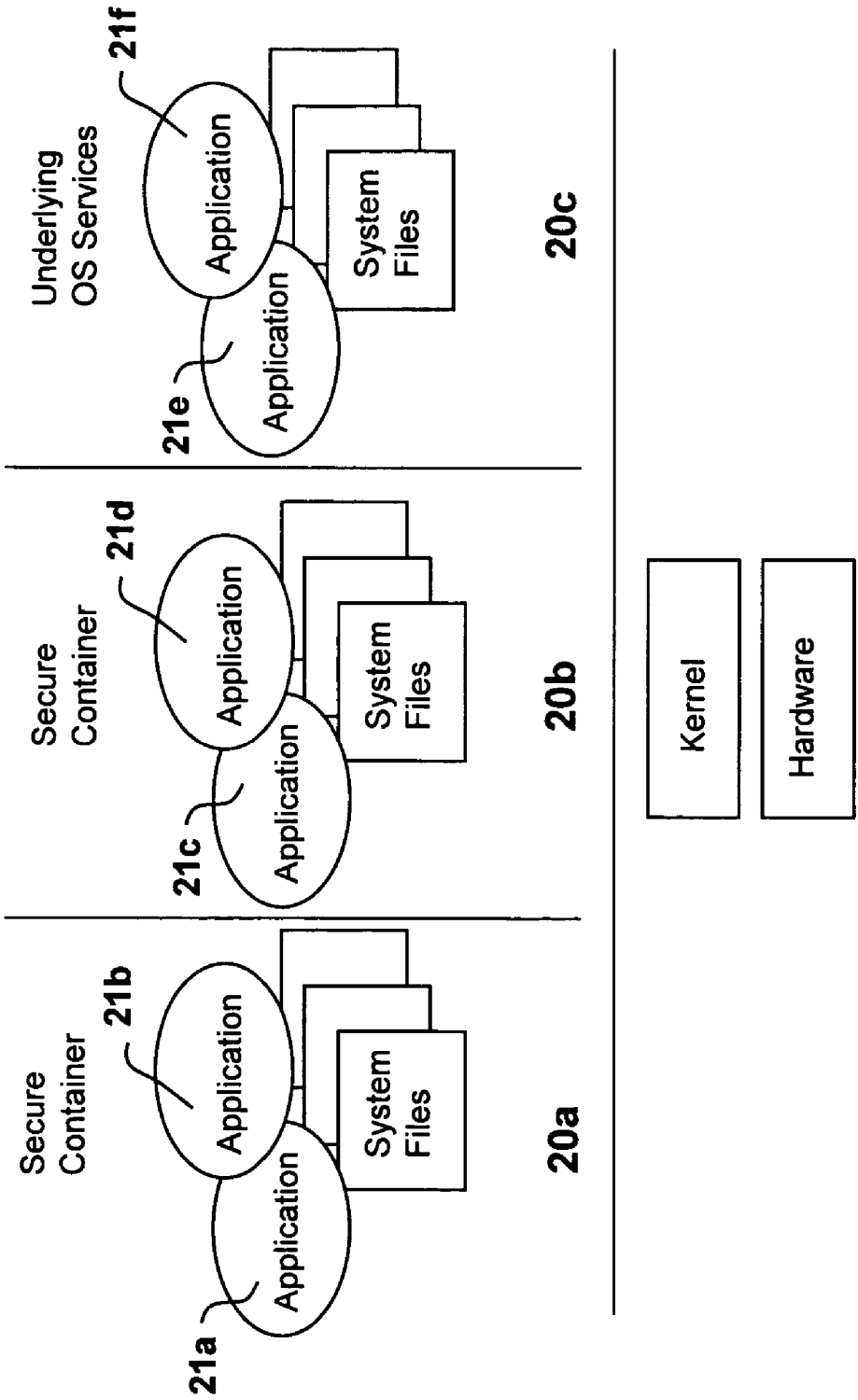


Figure 2

U.S. Patent

Apr. 14, 2009

Sheet 3 of 17

US 7,519,814 B2

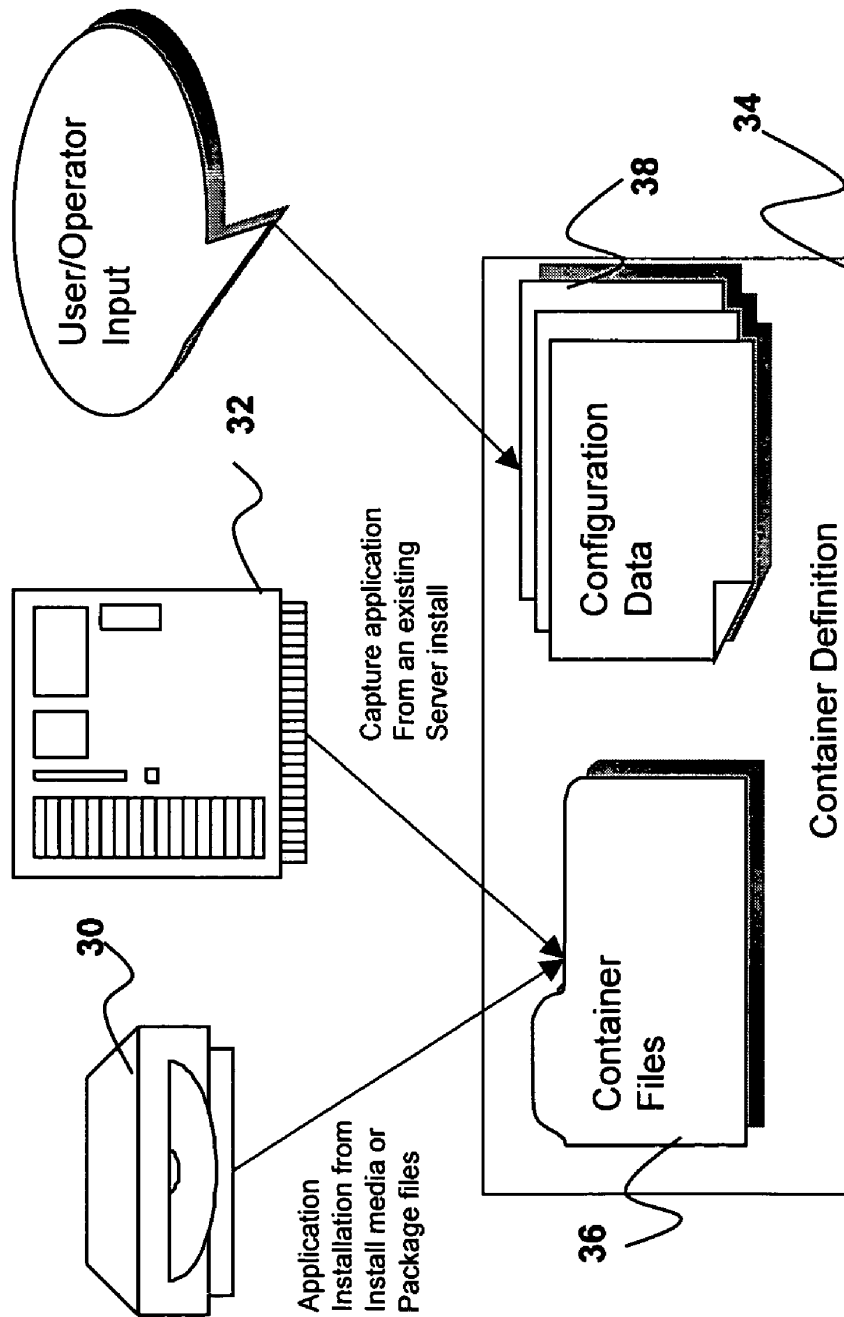
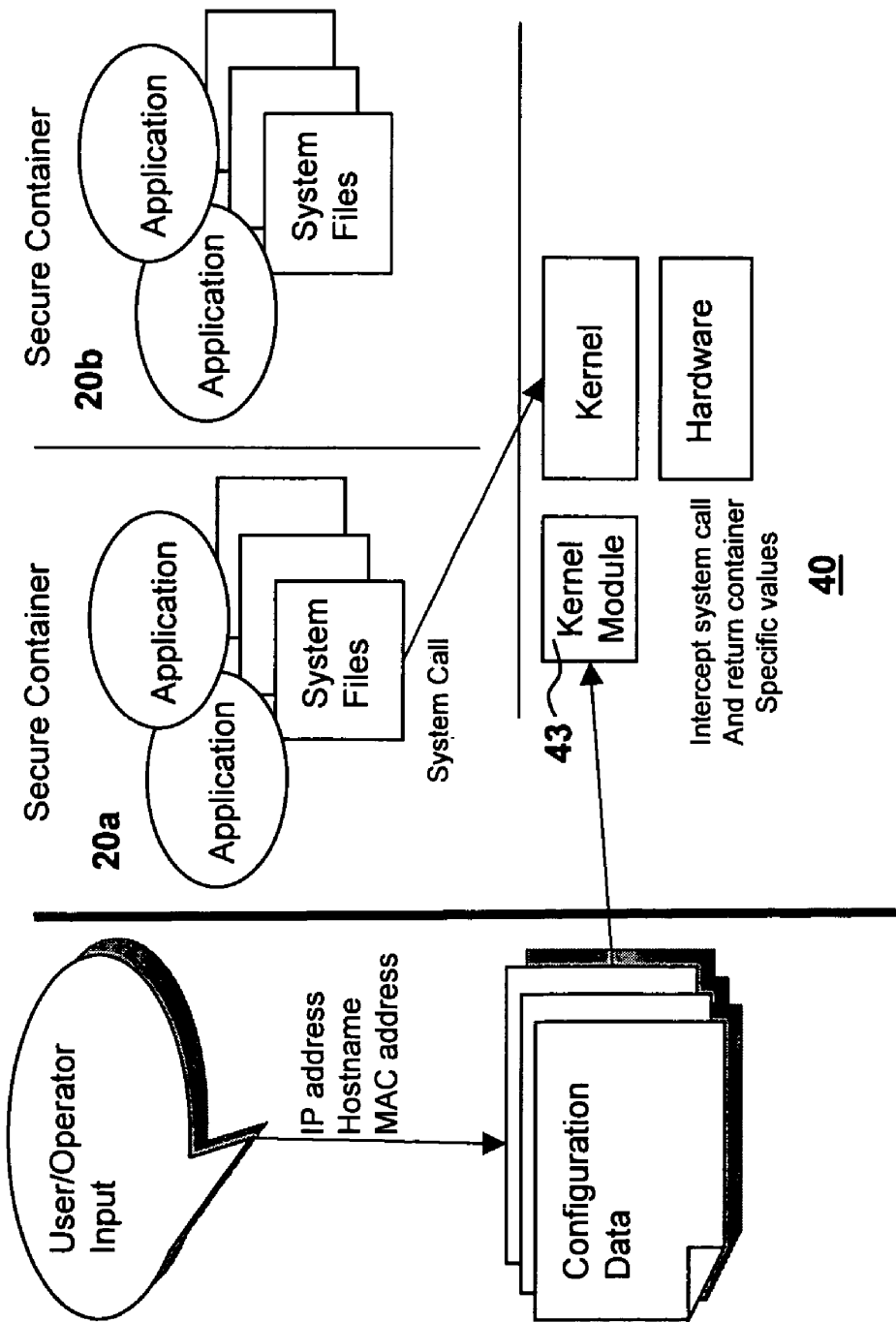


Figure 3



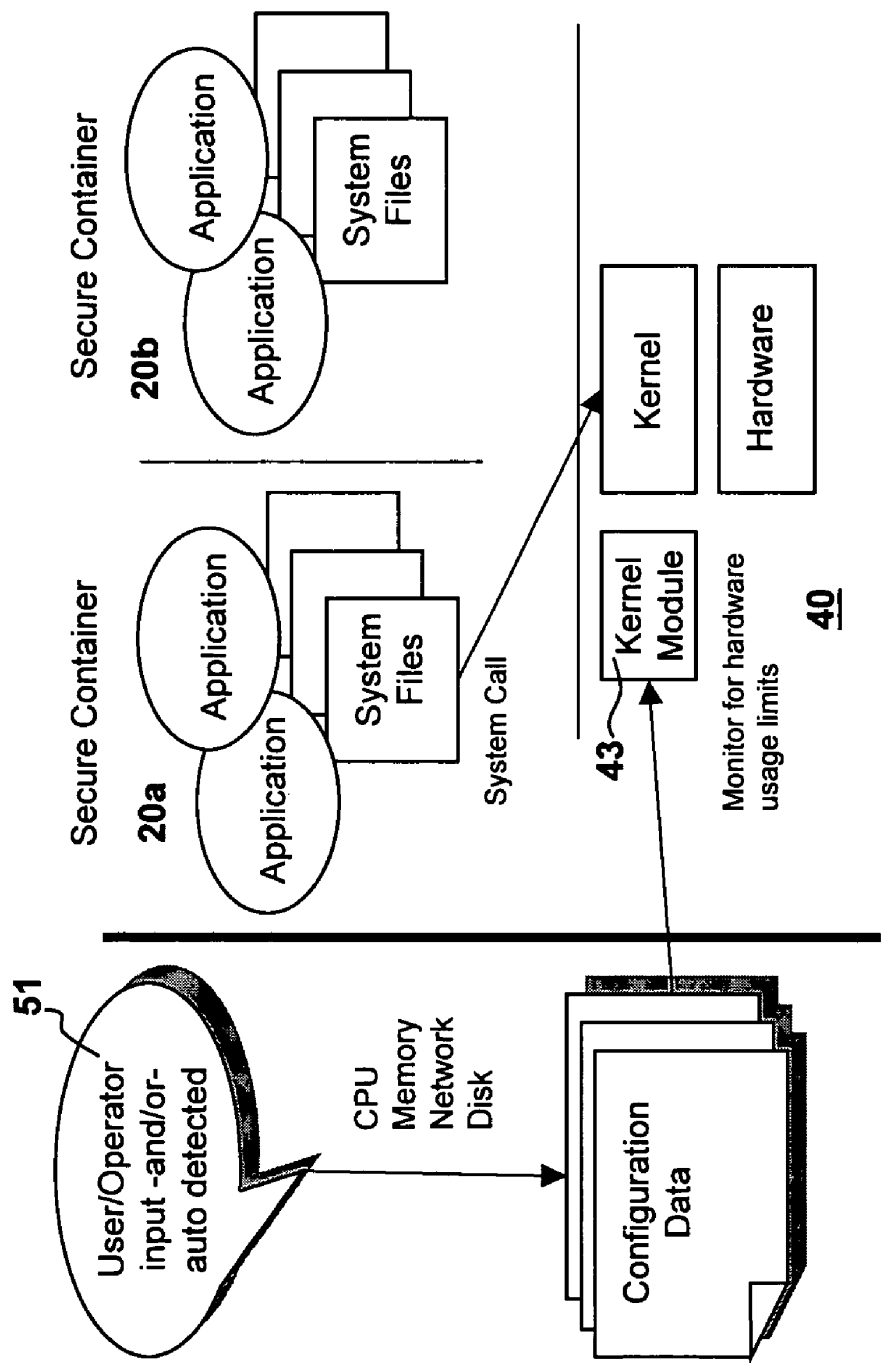


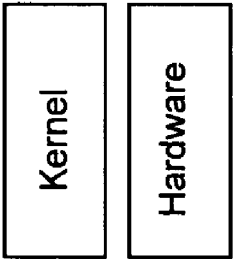
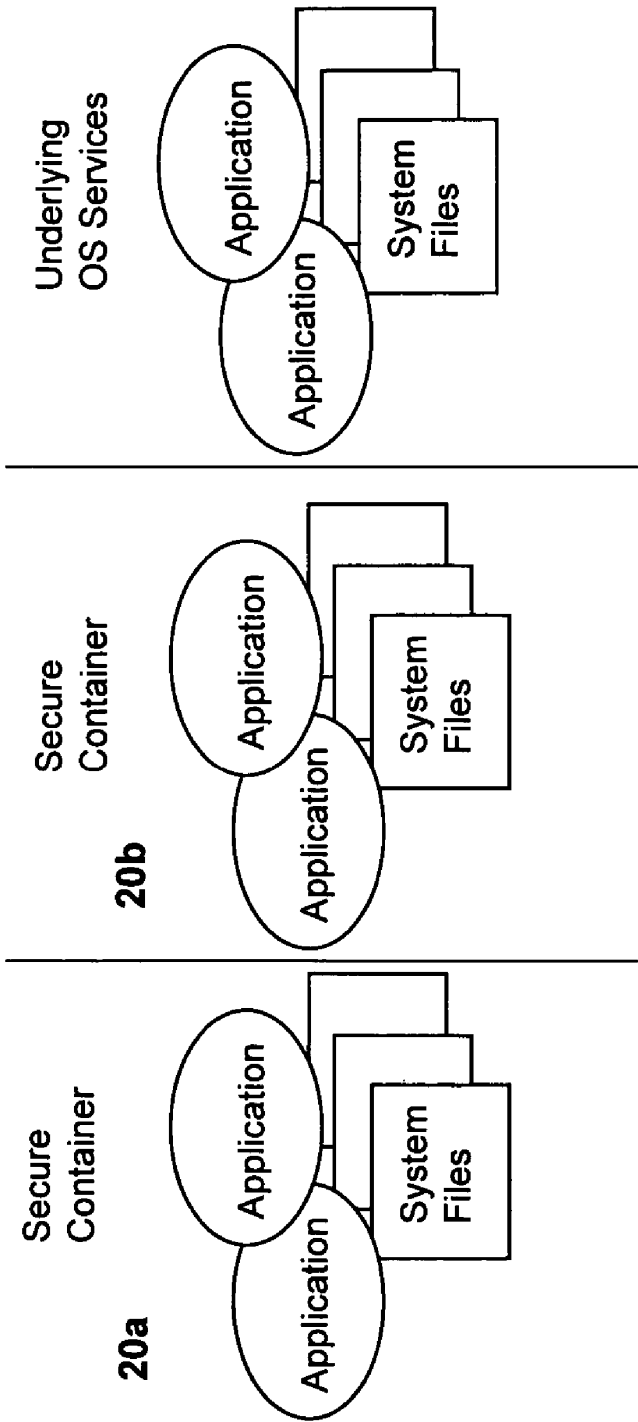
Figure 5

U.S. Patent

Apr. 14, 2009

Sheet 6 of 17

US 7,519,814 B2



40

Figure 6

U.S. Patent

Apr. 14, 2009

Sheet 7 of 17

US 7,519,814 B2

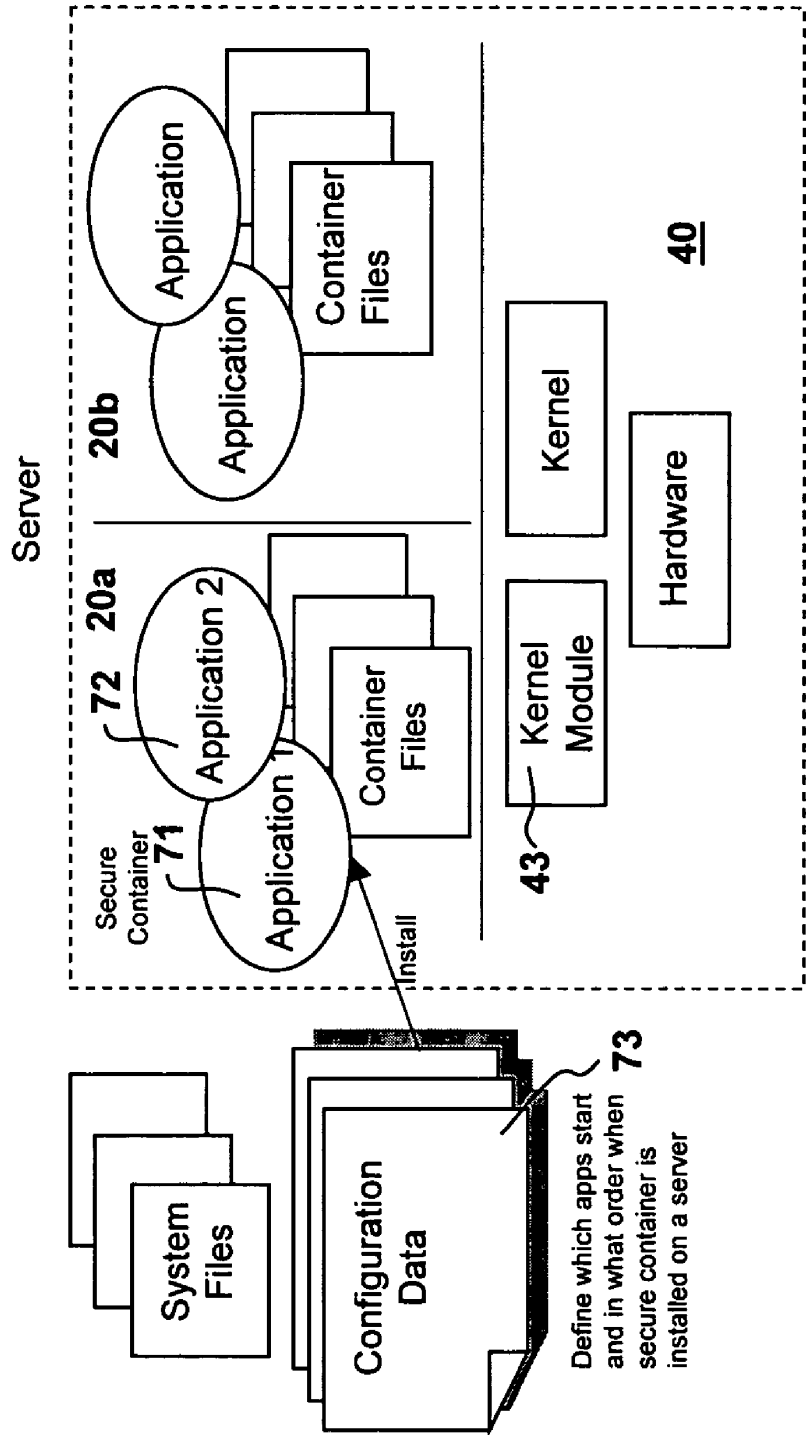


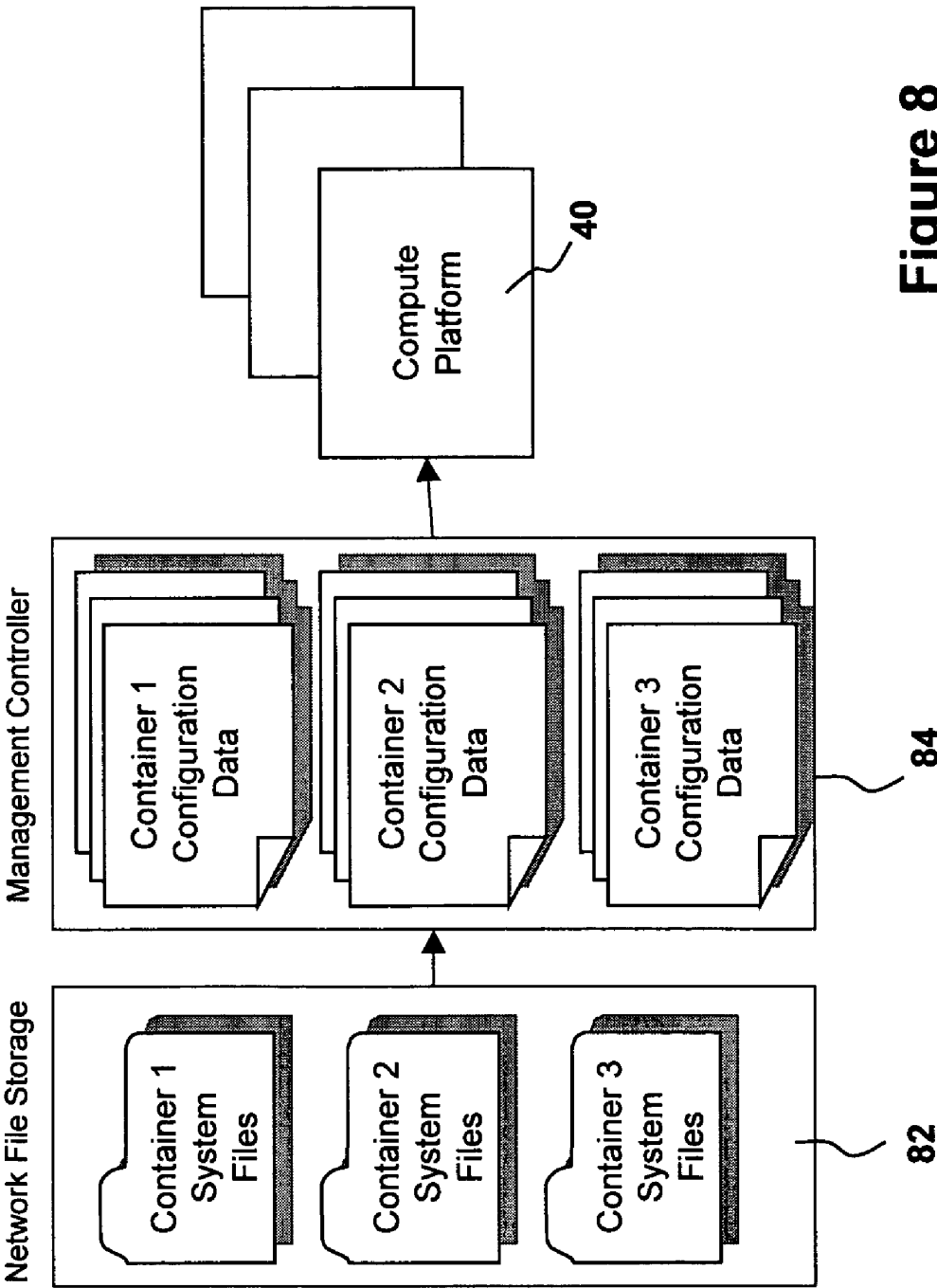
Figure 7

U.S. Patent

Apr. 14, 2009

Sheet 8 of 17

US 7,519,814 B2



U.S. Patent

Apr. 14, 2009

Sheet 9 of 17

US 7,519,814 B2

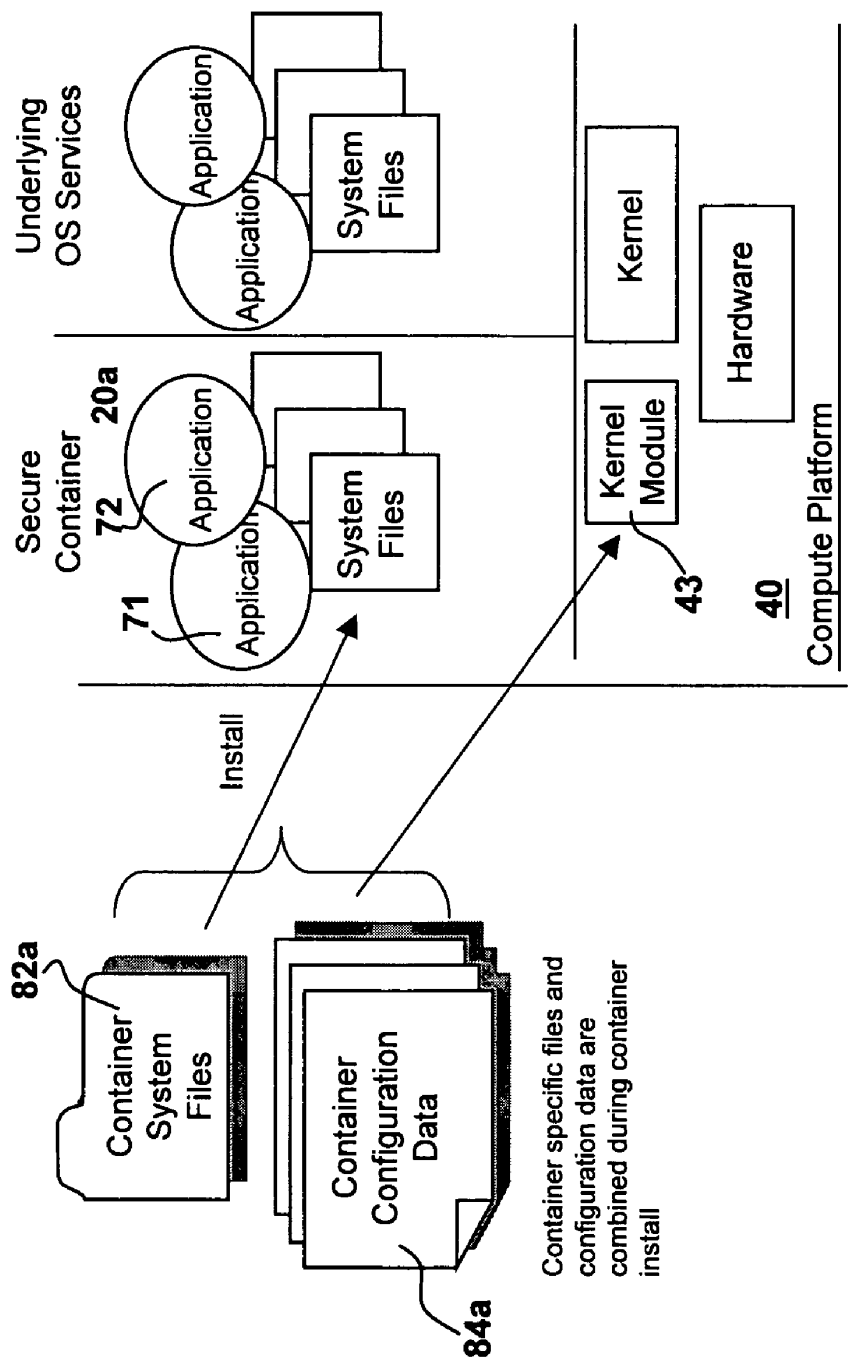


Figure 9

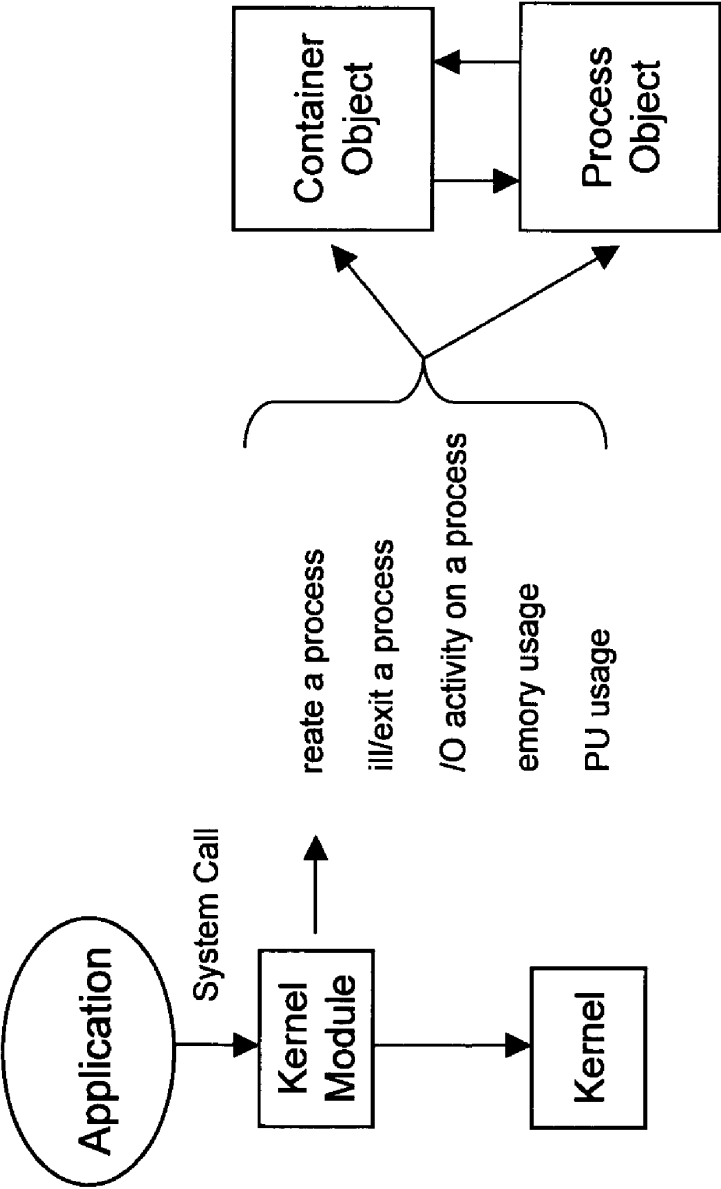


Figure 10

U.S. Patent

Apr. 14, 2009

Sheet 11 of 17

US 7,519,814 B2

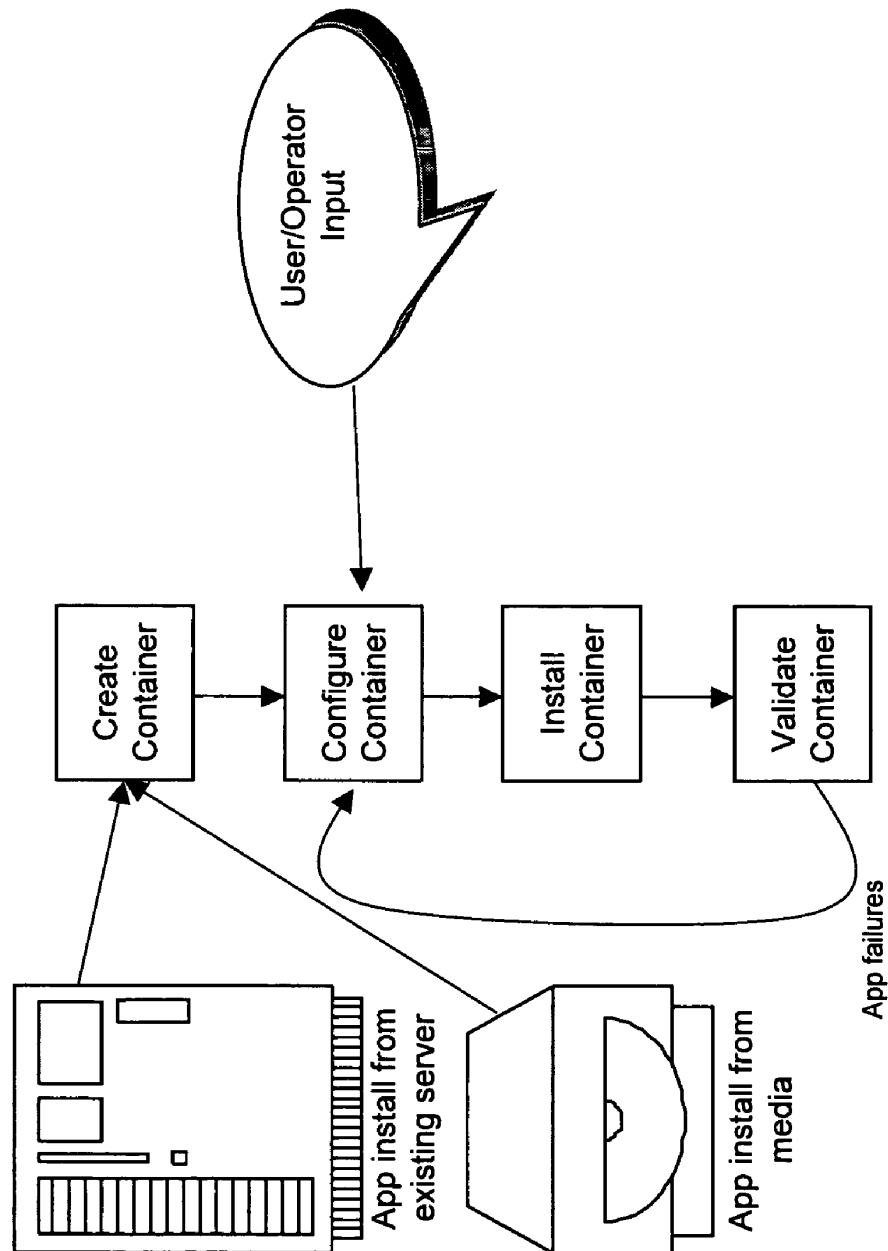


Figure 11

U.S. Patent

Apr. 14, 2009

Sheet 12 of 17

US 7,519,814 B2

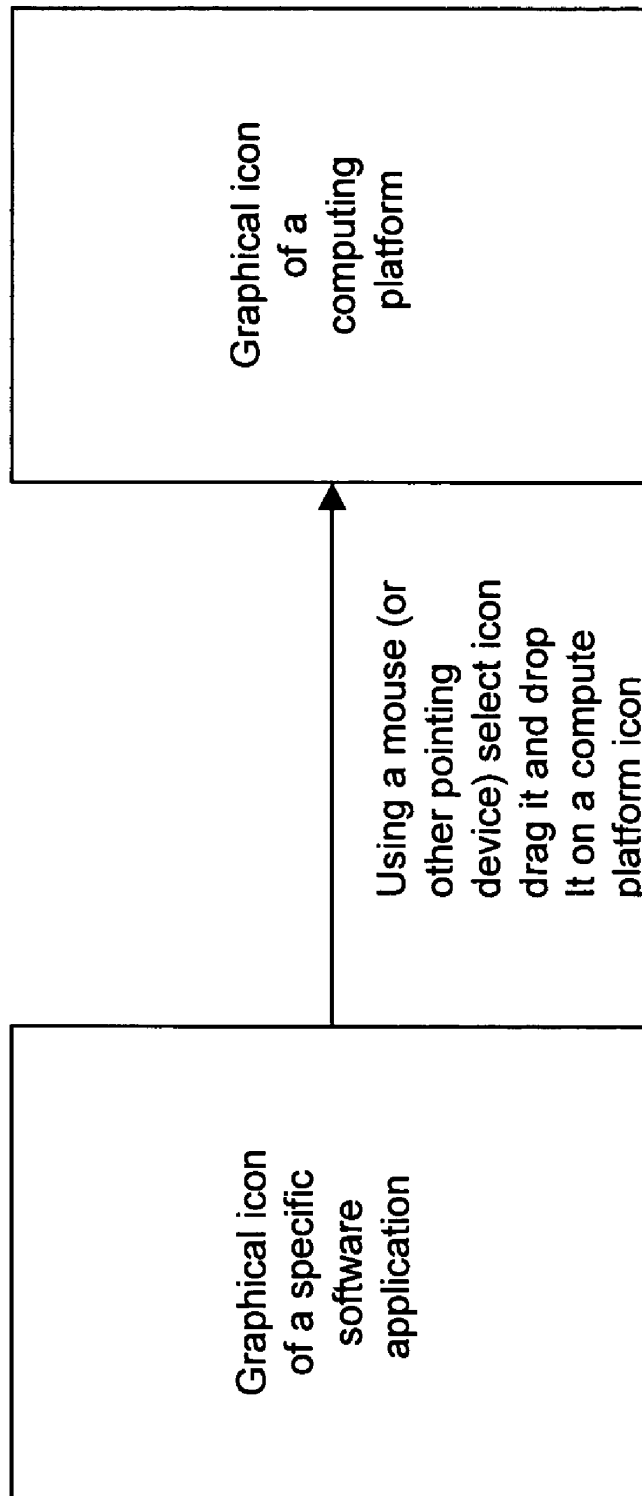


Figure 12

U.S. Patent

Apr. 14, 2009

Sheet 13 of 17

US 7,519,814 B2

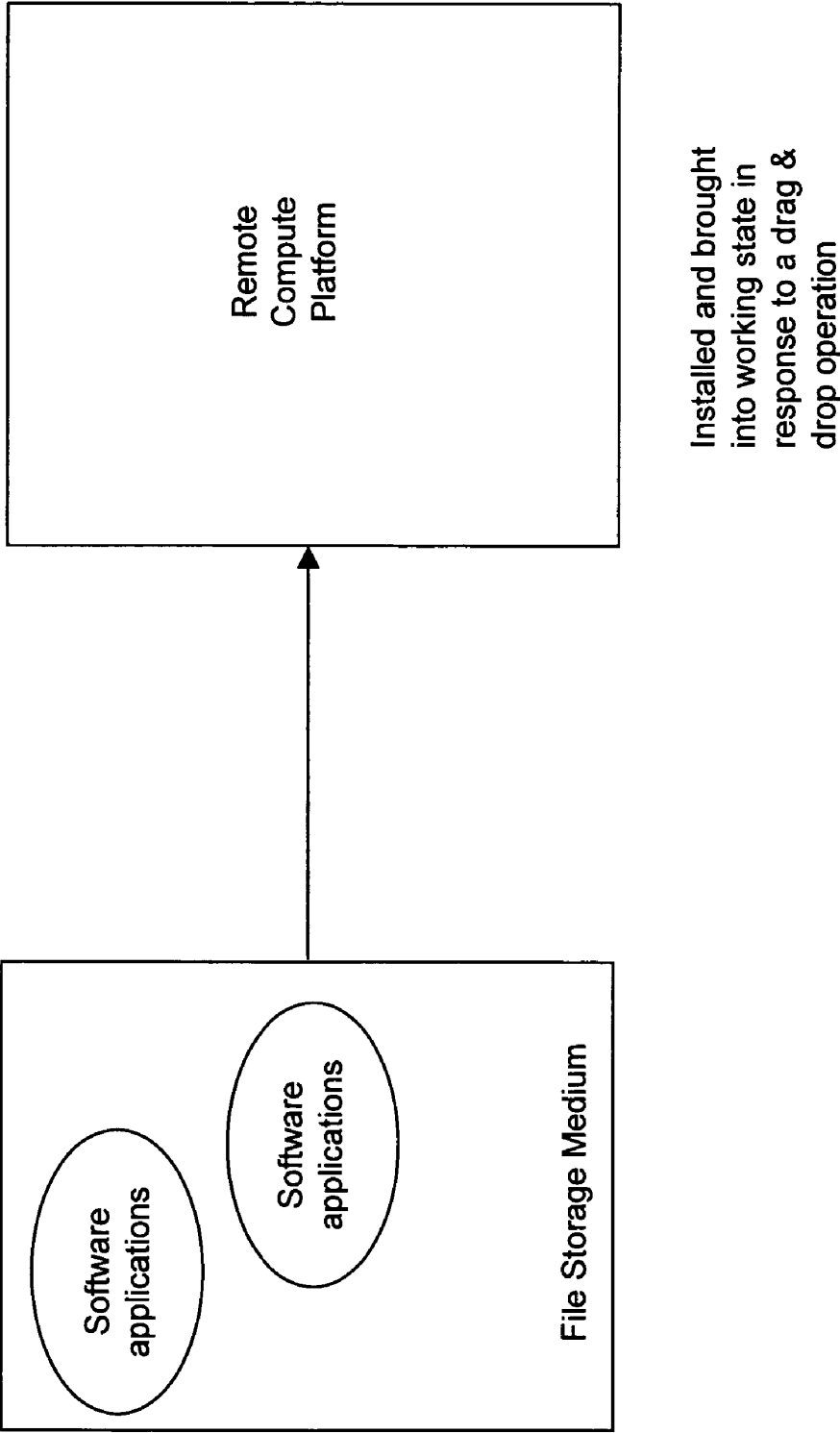


Figure 13

U.S. Patent

Apr. 14, 2009

Sheet 14 of 17

US 7,519,814 B2

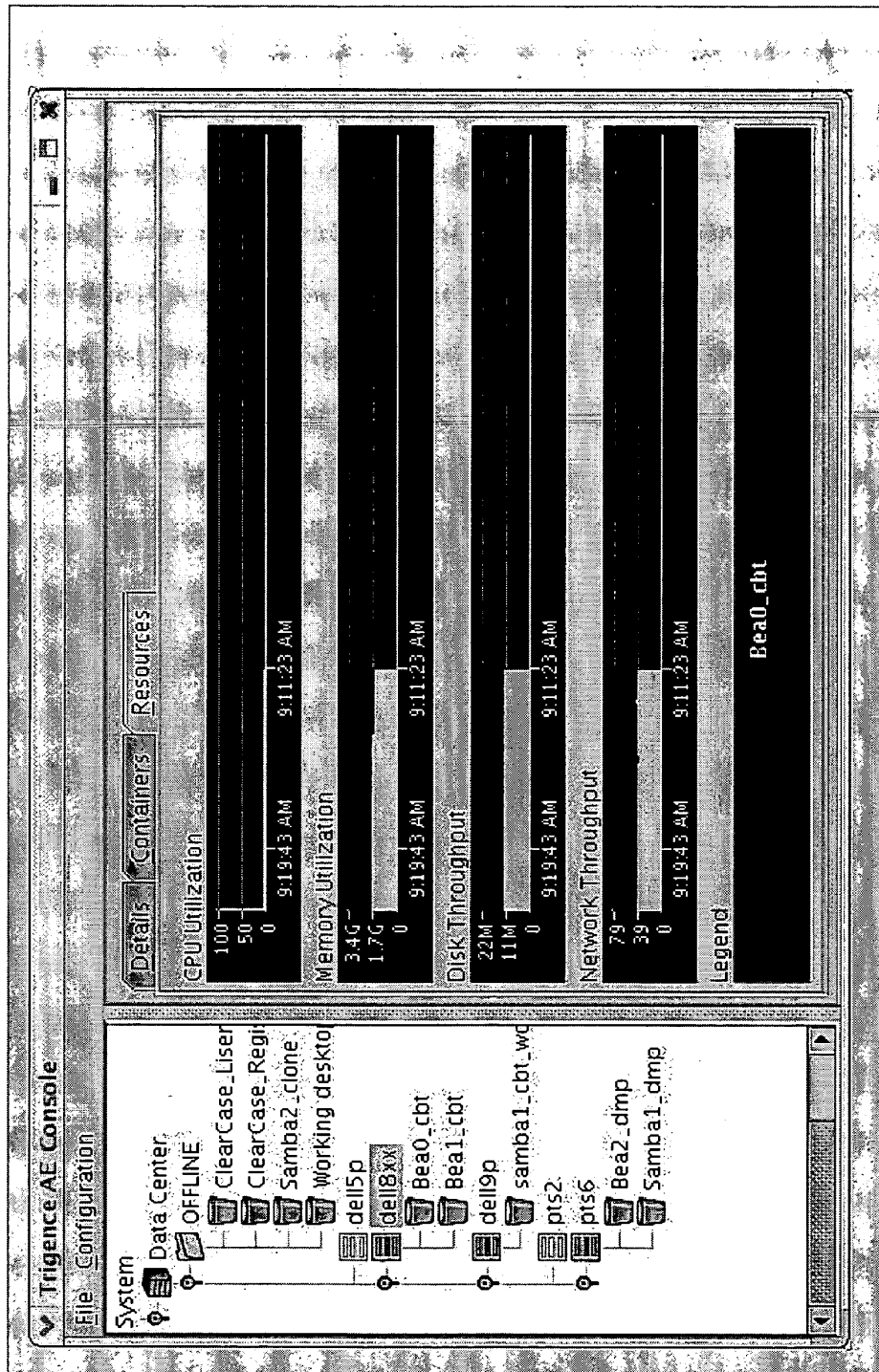


Figure 14

U.S. Patent

Apr. 14, 2009

Sheet 15 of 17

US 7,519,814 B2

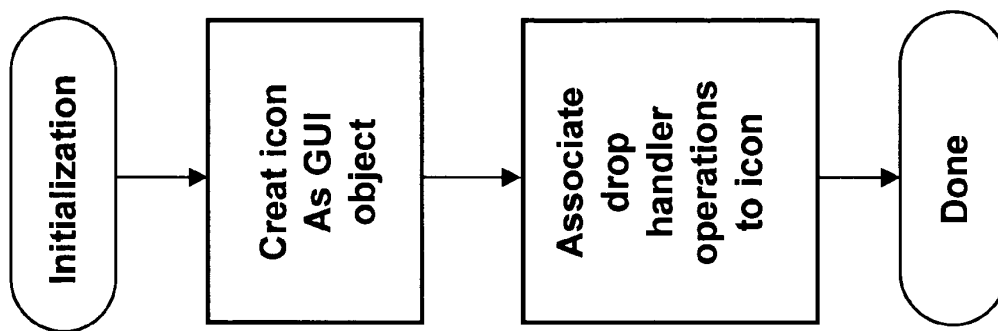


Figure 15

U.S. Patent

Apr. 14, 2009

Sheet 16 of 17

US 7,519,814 B2

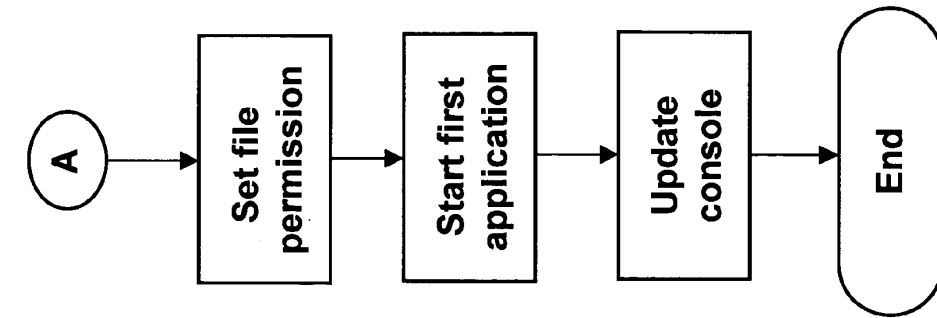
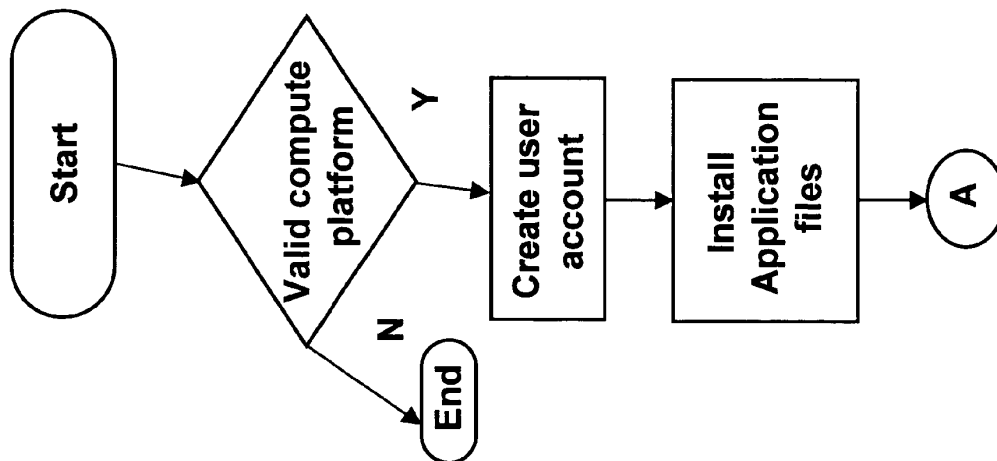


Figure 16



U.S. Patent

Apr. 14, 2009

Sheet 17 of 17

US 7,519,814 B2

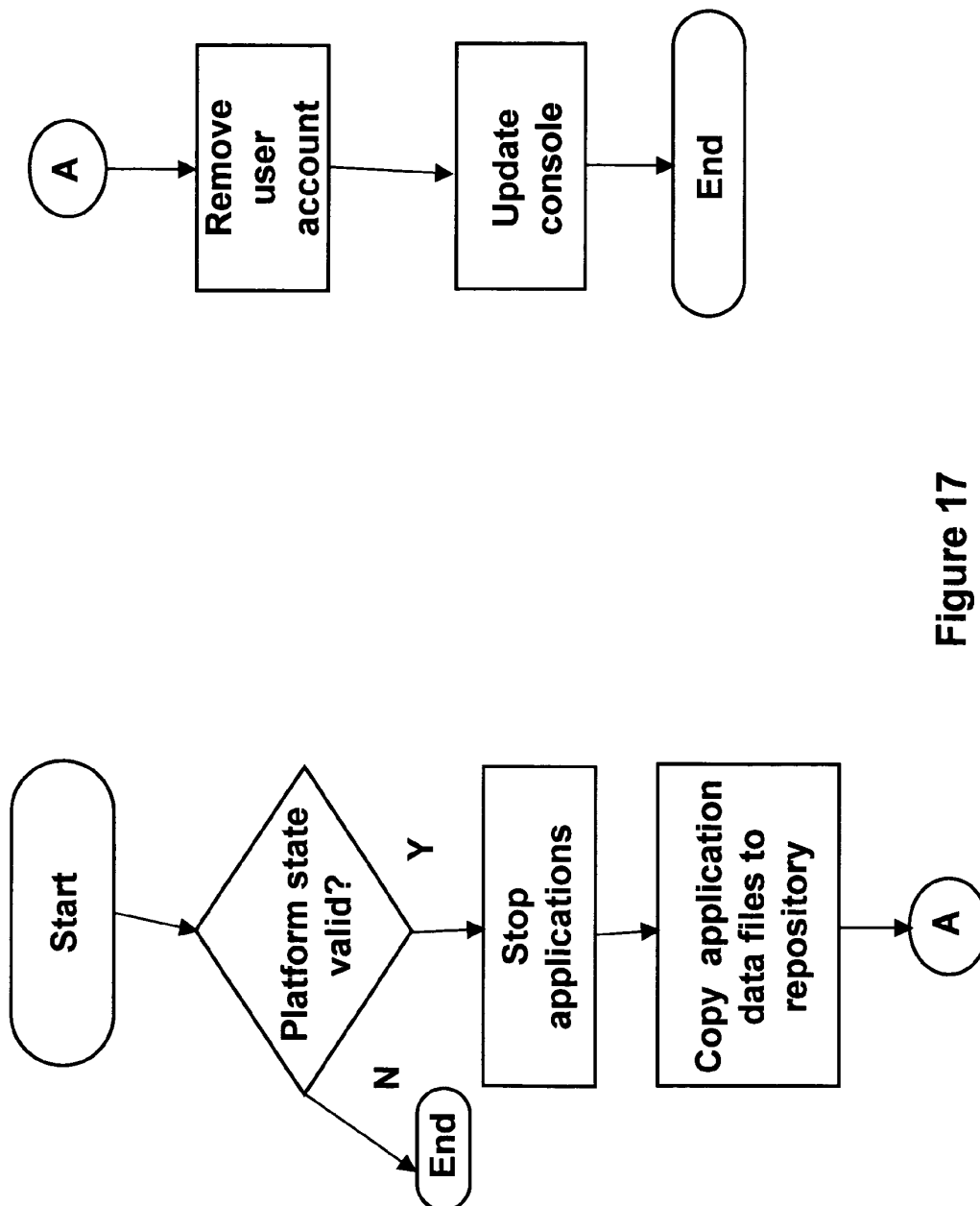


Figure 17

US 7,519,814 B2

1

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**CROSS-REFERENCE TO RELATED APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

2

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

```
/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1
```

US 7,519,814 B2

3

/lib/libnsl.so.1
 /lib/libdl.so.2
 /lib/i686/libpthread.so.0
 /lib/i686/libc.so.6
 /lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
 /etc/httpd/conf
 /etc/httpd/conf.d
 /etc/httpd/logs
 /etc/httpd/modules
 /etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

4

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySQL and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

US 7,519,814 B2

5

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of

6

this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed “spoofing” and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. 3 is a pictorial diagram illustrating the creation of a container.

FIG. 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the “kernel module” which is used to handle system calls.

FIG. 5 is a diagram similar to FIG. 4, wherein additional configuration data is provided.

FIG. 6 is a diagram illustrating a system wherein the containers are secure containers.

FIG. 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. 9 is a diagram that illustrates the installation of a container on a server.

FIG. 10 is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. 14 is a screen snapshot of an implementation according to the schematic of FIG. 13.

FIG. 15 is a flow chart of a method of initializing icons of FIG. 14.

FIG. 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. 14; and,

FIG. 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13.

DETAILED DESCRIPTION

Turning now to FIG. 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c.

US 7,519,814 B2

7

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. 2 illustrates a system in accordance with the invention in which multiple applications **21a**, **21b**, **21c**, **21d**, **21e**, and **21f** can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20a**, **20b** and **20c**. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20a**, **20b** and **20c** are shown in FIG. 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. 4 illustrates a system having two secure containers **20a** **20b**, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20a**, **20b** environment system calls are intercepted, by the kernel module **43** passing through services installed as part of

8

this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

As introduced heretofore, “spoofing” is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system’s kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the ‘uname’ system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has ‘spoofed’ the ‘uname’ system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same “spoofing” mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

- tracking applications, for example a tracking of which applications are in and out of a specific container;
- spoofing particular values returned by the kernel;
- applying resource checks and imposing resource limits;
- monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,
- retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20a** or **20b**. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. 6 that each application executing associated with a container **20a** or **20b**, or contained within a

US 7,519,814 B2

9

container **20a** or **20b**, is able to access files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20a** association are not able to access the files contained in another container **20b**.

When a container **20a** or **20b** is installed specific applications are started, as is shown in FIG. 7 and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. 7, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. 8, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. 9 shows that the files **82a**, **84a** are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing

10

the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. 9.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform and then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. 3, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

US 7,519,814 B2

11

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

- modifying the contents a file to add a container specific IP address;
- modifying the contents of a directory to define start scripts that should be executed;
- modifying the contents of a file to define which mount points will relate to a container;
- removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications

12

are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

US 7,519,814 B2

13

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

14

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

US 7,519,814 B2

15

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. 14, a screen snapshot of an implementation is shown according to the schematic of FIG. 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. 15, shown is a flow chart of a method of initializing the icons of FIG. 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. 16 are performed by an install drop handler. During installation, verification is performed to

16

determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. 15), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

US 7,519,814 B2

17

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

18

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

- a) running an instance of a service on a server;
- b) determining which files are being used; and,
- c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim 14 comprising the steps of:

- assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
- installing the container on a server; and,
- testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

- using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with a unique icon representing

US 7,519,814 B2

19

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim 20 further comprising starting a distributed software application.

22. A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim 17 further comprising creating a user account for the service.

25. A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim 1, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;
displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

20

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim 31, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

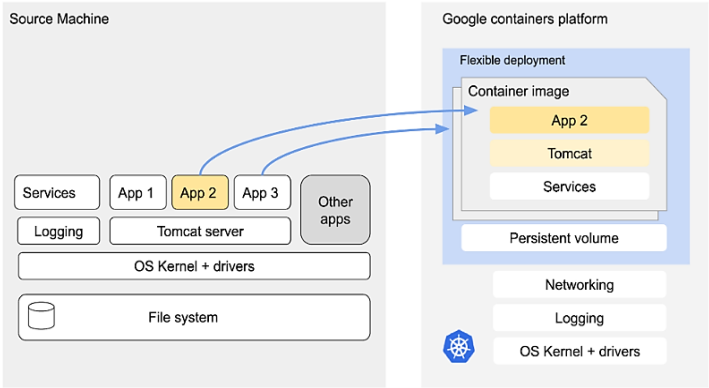
U.S. Patent No. 7,519,814 (“’814 Patent”)

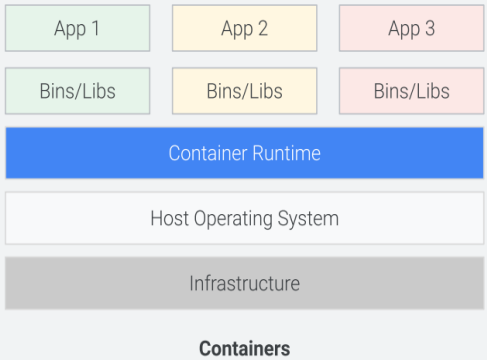
Accused Instrumentalities: Google’s “Migrate to Containers,” and all versions and variations thereof since the issuance of the asserted patent.


Claim 1

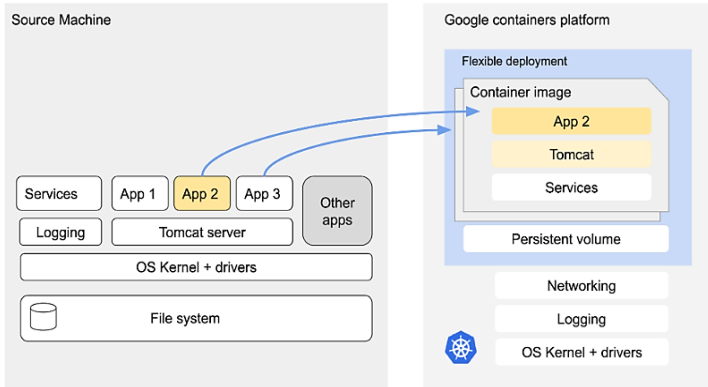
Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>To the extent the preamble is limiting, Google practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>Use Migrate to Containers to modernize traditional applications away from virtual machine (VM) instances and into native containers that run on Google Kubernetes Engine (GKE), Anthos clusters, or Cloud Run platform. You can migrate workloads from VMs that run on VMware or Compute Engine, giving you the flexibility to containerize your existing workloads with ease.</p> <p>https://cloud.google.com/migrate/containers/docs/getting-started, Last accessed on June 17, 2023</p> <p>Given that, using tools like Migrate to Containers is a uniquely smart, efficient way to modernize traditional applications away from virtual machines and into native containers. Our unique automation approach extracts critical application elements from a VM so you can easily insert those elements into containers running on Google Kubernetes Engine (GKE), without artifacts like guest OS layers that VMs need but that are unnecessary for containers.</p> <p>https://cloud.google.com/blog/products/containers-kubernetes/how-migrate-for-anthos-improves-vm-to-container-migration, Last accessed on June 17, 2023</p>

Claim 1	Accused Instrumentalities																		
	<p>Migrate to Containers supports migrations of VMs to containers on Google Kubernetes Engine on the 64-bit Linux operating systems listed in the following table.</p> <table><tr><th>OS</th><th>Compute Engine</th><th>VMware</th></tr><tr><td>CentOS</td><td>6.0, 7.0, 7.0 UEFI, 8.0</td><td>6.7, 6.9, 7.6</td></tr><tr><td>Debian</td><td>7.0, 8.0, 9.0, 10.0</td><td>9.4, 9.6</td></tr><tr><td>RHEL</td><td>6.0, 7.0, 7.0 UEFI, 7.4 SAP, 7.6 SAP, 8.0</td><td>6.5, 7.5, 7.6, 8.3</td></tr><tr><td>SUSE</td><td>12, 12 SP3 SAP, 12 SP4 SAP, 15, 15 SAP, 15 SP1 SAP</td><td>12 SP2, 12 SP3, 12 SP4, 15</td></tr><tr><td>Ubuntu</td><td>12 LTS, 14 LTS, 16 LTS, 16 LTS minimal, 18 LTS, 18 LTS minimal, 18 LTS UEFI, 19.04, 19.04 minimal</td><td>12.04.5 LTS, 14.04 LTS, 16.04 LTS, 18.04.10 LTS</td></tr></table> <p>https://cloud.google.com/migrate/containers/docs/compatible-os-versions, Last accessed on June 05, 2023</p> <p>Containers can run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or on physical servers; on a developer’s machine or in data centers on-premises; and of course, in the public cloud.</p> <p>https://cloud.google.com/learn/what-are-containers, Last accessed on June 17, 2023</p>	OS	Compute Engine	VMware	CentOS	6.0, 7.0, 7.0 UEFI, 8.0	6.7, 6.9, 7.6	Debian	7.0, 8.0, 9.0, 10.0	9.4, 9.6	RHEL	6.0, 7.0, 7.0 UEFI, 7.4 SAP, 7.6 SAP, 8.0	6.5, 7.5, 7.6, 8.3	SUSE	12, 12 SP3 SAP, 12 SP4 SAP, 15, 15 SAP, 15 SP1 SAP	12 SP2, 12 SP3, 12 SP4, 15	Ubuntu	12 LTS, 14 LTS, 16 LTS, 16 LTS minimal, 18 LTS, 18 LTS minimal, 18 LTS UEFI, 19.04, 19.04 minimal	12.04.5 LTS, 14.04 LTS, 16.04 LTS, 18.04.10 LTS
OS	Compute Engine	VMware																	
CentOS	6.0, 7.0, 7.0 UEFI, 8.0	6.7, 6.9, 7.6																	
Debian	7.0, 8.0, 9.0, 10.0	9.4, 9.6																	
RHEL	6.0, 7.0, 7.0 UEFI, 7.4 SAP, 7.6 SAP, 8.0	6.5, 7.5, 7.6, 8.3																	
SUSE	12, 12 SP3 SAP, 12 SP4 SAP, 15, 15 SAP, 15 SP1 SAP	12 SP2, 12 SP3, 12 SP4, 15																	
Ubuntu	12 LTS, 14 LTS, 16 LTS, 16 LTS minimal, 18 LTS, 18 LTS minimal, 18 LTS UEFI, 19.04, 19.04 minimal	12.04.5 LTS, 14.04 LTS, 16.04 LTS, 18.04.10 LTS																	

Claim 1	Accused Instrumentalities
	<p>A container is a way of packaging a given application's code and dependencies so that the application will run easily in any computing environment. This solves the common problem of https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p>  <p>The diagram illustrates the migration of an application from a Source Machine to the Google containers platform. On the left, the 'Source Machine' contains a stack of components: 'File system' at the base, followed by 'OS Kernel + drivers', 'Logging', 'Tomcat server', and 'Services'. Above these are 'App 1', 'App 2' (highlighted in yellow), 'App 3', and 'Other apps'. Two blue arrows point from 'App 2' to the 'Google containers platform' on the right. The platform consists of a 'Flexible deployment' layer containing a 'Container image' (with 'App 2' and 'Tomcat' inside) and 'Services'. Below this is a 'Persistent volume' layer, followed by 'Networking', 'Logging', and 'OS Kernel + drivers' at the base. A Google Cloud logo is visible at the bottom left of the platform stack.</p> <p>https://cloud.google.com/blog/products/application-modernization/shift-your-apps-to-container-based-workloads-on-the-command-line, Last accessed on June 17, 2023</p>

Claim 1	Accused Instrumentalities
	 <p>The diagram illustrates a container architecture stack. At the top, three application boxes labeled 'App 1' (green), 'App 2' (yellow), and 'App 3' (pink) are shown. Below each app is a corresponding 'Bins/Libs' box in the same color. These are all contained within a blue 'Container Runtime' box. This runtime sits on top of a light gray 'Host Operating System' box, which in turn sits on a dark gray 'Infrastructure' box. A light gray box labeled 'Containers' is positioned at the bottom of the stack, encompassing the runtime, OS, and infrastructure layers.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p> <p>Containers virtualize CPU, memory, storage, and network resources at the operating system level, providing developers with a view of the OS logically isolated from other applications.</p> <p>https://cloud.google.com/learn/what-are-containers, Last accessed on June 17, 2023</p> <p>Containers are much more lightweight than VMs</p> <p>Containers virtualize at the OS level while VMs virtualize at the hardware level</p> <p>Containers share the OS kernel and use a fraction of the memory VMs require</p> <p>https://cloud.google.com/learn/what-are-containers, Last accessed on June 17, 2023</p>

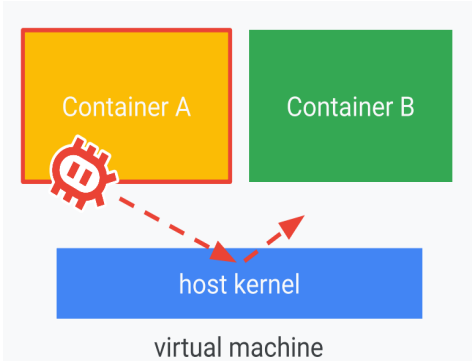
Claim 1	Accused Instrumentalities
	<p>Containers use specific features of the Linux kernel that “trick” individual applications into thinking they’re in their own unique environment, even though multiple applications share the same host kernel. (If you’re not familiar with the Linux kernel, it’s a part of the operating system that communicates between processes--requests that do user tasks like opening a file, running a program-- and the hardware. It manages resources like memory and CPU to meet these requests).</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p> <p>The core components of the Linux kernel that are used for containers are cgroups — control groups, which define the resources like CPU and memory which are available to a given process — and namespaces, which are a way of separating processes by restricting what each process can see, so that system resources “appear” isolated to the process.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p>
<p>[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>The method practiced by Google through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p> <p><i>See, e.g.:</i></p> <p>There are several storage options for applications running on Google Kubernetes Engine (GKE). The choices vary in terms of</p> <p>Volumes are a storage unit accessible to containers in a Pod. Some volume types are backed by ephemeral storage. Ephemeral storage types (for example, emptyDir ) do not persist after the Pod ceases to exist. These types are useful for scratch space for applications. You can manage your local ephemeral storage resources as you do your CPU and memory resources. Other volume types are backed by durable storage.</p>

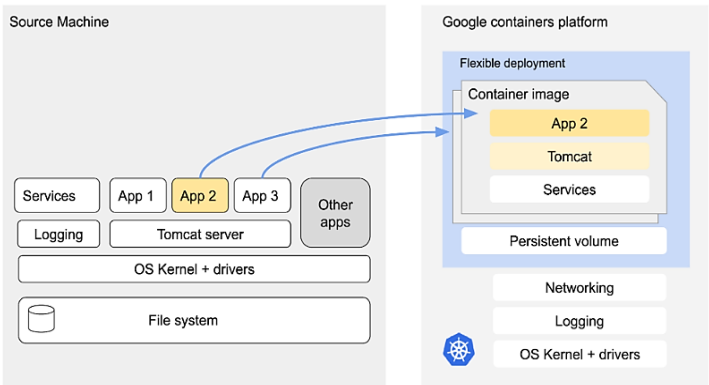
Claim 1	Accused Instrumentalities
	<p>https://cloud.google.com/kubernetes-engine/docs/concepts/storage-overview, Last accessed on June 17, 2023</p> <p>At its core, a volume is a directory, possibly with some data in it, which is accessible to the containers in a pod. How that directory comes to be, the</p> <p><code>.spec.containers[*].volumeMounts</code>. A process in a container sees a filesystem view composed from the initial contents of the container image, plus volumes (if defined) mounted inside the container. The process sees a root filesystem that initially matches the contents of the container image. Any writes to within that filesystem hierarchy, if allowed, affect what that process views when it performs a subsequent filesystem access. Volumes mount at the specified paths within the image. For each container defined within a Pod, you must independently specify where to mount each volume that the container uses.</p> <p>https://kubernetes.io/docs/concepts/storage/volumes/, Last accessed on June 17, 2023</p>  <p>https://cloud.google.com/blog/products/application-modernization/shift-your-apps-to-container-based-workloads-on-the-command-line, Last accessed on June 17, 2023</p>

Claim 1	Accused Instrumentalities
	<p>A container is a way of packaging a given application's code and dependencies so that the application will run easily in any computing environment. This solves the common problem of</p> <p>The container image specifies the container's file system. For example, if you're running a Node.js application, the container image would contain your app, Node.js, and other dependencies like Linux system libraries (except the kernel). A container image usually extends a base operating system image, or base image. This base image is the basis of your container, so you'll want to ensure that it's properly patched and free from known vulnerabilities.</p> <p>workloads onto each server. As such, the architecture of containers means that they're deployed with multiple containers sharing the same kernel.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p> <p>Containers are lightweight packages of your application code together with dependencies such as specific versions of programming language runtimes and libraries required to run your software services.</p> <p>https://cloud.google.com/learn/what-are-containers, Last accessed on June 17, 2023</p>
[1b] wherein the set of associated system files are compatible with a local kernel	In the method practiced by Google through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.

Claim 1	Accused Instrumentalities
of at least some of the plurality of different operating systems,	<p data-bbox="651 321 766 354"><i>See, e.g.:</i></p> <p data-bbox="651 394 1711 548">A container is a way of packaging a given application's code and dependencies so that the application will run easily in any computing environment. This solves the common problem of</p> <p data-bbox="651 589 1501 873">The container image specifies the container's file system. For example, if you're running a Node.js application, the container image would contain your app, Node.js, and other dependencies like Linux system libraries (except the kernel). A container image usually extends a base operating system image, or base image. This base image is the basis of your container, so you'll want to ensure that it's properly patched and free from known vulnerabilities.</p> <p data-bbox="651 881 1669 1027">Containers use specific features of the Linux kernel that "trick" individual applications into thinking they're in their own unique environment, even though multiple applications share the same host kernel. (If you're not familiar with the Linux kernel, it's a part of the operating system that communicates between processes--requests that do user tasks like opening a file, running a program-- and the hardware. It manages resources like memory and CPU to meet these requests).</p> <p data-bbox="651 1036 1879 1101">https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p>

Claim 1	Accused Instrumentalities
	<p>Containers can run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or on physical servers; on a developer's machine or in data centers on-premises; and of course, in the public cloud.</p> <p>https://cloud.google.com/learn/what-are-containers, Last accessed on June 17, 2023</p>
<p>[1c] the containers of application software excluding a kernel,</p>	<p>In the method practiced by Google through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p><i>See, e.g.:</i></p> <ul style="list-style-type: none"> • Higher utilization and density, leveraging automatic bin-packing and auto-scaling capabilities, Kubernetes places containers optimally in nodes based on required resources while scaling as needed, without impairing availability. In addition, unlike VMs, all containers on a single node share one copy of the operating system and don't each require their own OS image and vCPU, resulting in a much smaller memory footprint and CPU needs. This means more workloads running on fewer compute resources. <p>https://cloud.google.com/blog/products/containers-kubernetes/how-migrate-for-anthos-improves-vm-to-container-migration, Last accessed on June 17, 2023</p>

Claim 1	Accused Instrumentalities
	<p>workloads onto each server. As such, the architecture of containers means that they're deployed with multiple containers sharing the same kernel.</p>  <p>The diagram illustrates a virtual machine environment. Inside the virtual machine, there are two containers: Container A (orange) and Container B (green). Both containers are connected to a shared host kernel (blue box) via dashed red arrows. A red gear icon with a dollar sign is positioned near Container A, indicating a security or financial aspect. The entire setup is labeled 'virtual machine' below the host kernel box.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p>
[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,	<p>In the method practiced by Google through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>One of the primary reasons to adopt containers is for your applications to be decoupled from the underlying environment and support higher resource utilization by “bin packing” multiple workloads onto each server. As such, the architecture of containers means that they’re deployed with multiple containers sharing the same kernel.</p> <p>The container image specifies the container’s file system. For example, if you’re running a Node.js application, the container image would contain your app, Node.js, and other dependencies like Linux system libraries (except the kernel). A container image usually extends a base operating system image, or base image. This base image is the basis of your container, so you’ll want to ensure that it’s properly patched and free from known vulnerabilities.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p>  <p>The diagram illustrates the architectural shift from a traditional source machine to a containerized environment. On the left, the 'Source Machine' shows a stack where applications (App 1, App 2, App 3) and services are layered on top of a Tomcat server, which runs on the OS Kernel + drivers, all within a File system. On the right, the 'Google containers platform' shows a 'Flexible deployment' model. Here, a 'Container image' (containing App 2, Tomcat, and Services) is deployed on a 'Persistent volume' layer, which sits above 'Networking', 'Logging', and the 'OS Kernel + drivers'. Blue arrows indicate the migration of App 2 and its dependencies from the source machine to the container image in the Google platform.</p> <p>https://cloud.google.com/blog/products/application-modernization/shift-your-apps-to-container-based-workloads-on-the-command-line, Last accessed on June 17, 2023</p>

Claim 1	Accused Instrumentalities
<p>[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by Google through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p><i>See, e.g.:</i></p> <p>One of the primary reasons to adopt containers is for your applications to be decoupled from the underlying environment and support higher resource utilization by “bin packing” multiple workloads onto each server. As such, the architecture of containers means that they’re deployed with multiple containers sharing the same kernel.</p> <p>The container image specifies the container’s file system. For example, if you’re running a Node.js application, the container image would contain your app, Node.js, and other dependencies like Linux system libraries (except the kernel). A container image usually extends a base operating system image, or base image. This base image is the basis of your container, so you’ll want to ensure that it’s properly patched and free from known vulnerabilities.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p>
<p>[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,</p>	<p>In the method practiced by Google through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containers use specific features of the Linux kernel that “trick” individual applications into thinking they’re in their own unique environment, even though multiple applications share the same host kernel. (If you’re not familiar with the Linux kernel, it’s a part of the operating system that communicates between processes--requests that do user tasks like opening a file, running a program-- and the hardware. It manages resources like memory and CPU to meet these requests).</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p> <p>The core components of the Linux kernel that are used for containers are cgroups — control groups, which define the resources like CPU and memory which are available to a given process — and namespaces, which are a way of separating processes by restricting what each process can see, so that system resources “appear” isolated to the process.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf, Last accessed on June 17, 2023</p> <p>reason. Furthermore, files within a container are inaccessible to other containers running in the same Pod [link]. The Kubernetes</p> <p>https://cloud.google.com/kubernetes-engine/docs/concepts/volumes, Last accessed on June 17, 2023</p> <p>A <i>Pod</i> (as in a pod of whales or pea pod) is a group of one or more <u>containers</u>, with shared storage and network resources, and a specification for how to run the containers. A Pod's contents are always co-located and co-scheduled, and run in a shared context. A Pod models an application-specific "logical host": it contains one or more application containers which are relatively tightly coupled. In non-cloud contexts, applications executed on the same physical or virtual machine are analogous to cloud applications executed on the same logical host.</p> <p>The shared context of a Pod is a set of Linux namespaces, cgroups, and potentially other facets of isolation - the same things that isolate a container. Within a Pod's context, the individual applications may have further sub-isolations applied.</p> <p>https://kubernetes.io/docs/concepts/workloads/pods/, Last accessed on June 17, 2023</p>

Claim 1	Accused Instrumentalities
	<p>ranges can access. GKE Sandbox for the Standard mode of operation provides a second layer of defense between containerized workloads on GKE for enhanced workload security. GKE https://cloud.google.com/kubernetes-engine#section-2, Last accessed on June 17, 2023</p>
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by Google through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p><i>See, e.g.:</i></p> <p>The original purpose of the cgroup, chroot, and namespace facilities in the kernel was to protect applications from noisy, nosey, and messy neighbors. Combining these with container images created an abstraction that also isolates applications from the (heterogeneous) operating systems on which they run. This decoupling of image and OS makes it possible to provide the same deployment environment in both development and production, which, in turn, improves deployment reliability and speeds up development by reducing inconsistencies and friction.</p> <p>“Borg, Omega, and, Kubernetes,” https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/44843.pdf, Last accessed on June 17, 2023</p>

Claim 1	Accused Instrumentalities
	<p>In Docker and Kubernetes, the container's root filesystem (rootfs) is based on the filesystem packaged with the image. The image's filesystem is immutable. Any change a container makes to the rootfs is stored separately and is destroyed with the container. This way, the image's filesystem https://opensource.googleblog.com/2023/04/gvisor-improves-performance-with-root-filesystem-overlay.html, Last accessed on June 17, 2023</p> <p>To use a volume, specify the volumes to provide for the Pod in <code>.spec.volumes</code> and declare where to mount those volumes into containers in <code>.spec.containers[*].volumeMounts</code>. A process in a container sees a filesystem view composed from the initial contents of the <u>container image</u>, plus volumes (if defined) mounted inside the container. The process sees a root filesystem that initially matches the contents of the container image. Any writes to within that filesystem hierarchy, if allowed, affect what that process views when it performs a subsequent filesystem access. Volumes mount at the <u>specified paths</u> within the image. For each container defined within a Pod, you must independently specify where to mount each volume that the container uses.</p> <p>https://kubernetes.io/docs/concepts/storage/volumes/, Last accessed on June 17, 2023</p>

Exhibit 3



US007784058B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,784,058 B2**
(45) **Date of Patent:** **Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES**

2002/0004854 A1 1/2002 Hartley
2002/0174215 A1 11/2002 Schaefer 709/224
2003/0101292 A1 5/2003 Fisher
2004/0025165 A1* 2/2004 Desoli et al. 719/310

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul
O’Leary**, Kanata (CA); **Dean Huffman**,
Kanata (CA)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002
WO WO 2006/039181 A 4/2006

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

OTHER PUBLICATIONS

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 1293 days.

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

* cited by examiner

(21) Appl. No.: **10/946,536**

Primary Examiner—Hyung S Sough

Assistant Examiner—Syed Roni

(22) Filed: **Sep. 21, 2004**

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt,
Milbrath & Gilchrist, P.A. Attorneys at Law

(65) **Prior Publication Data**
US 2005/0066303 A1 Mar. 24, 2005

(57) **ABSTRACT**

Related U.S. Application Data

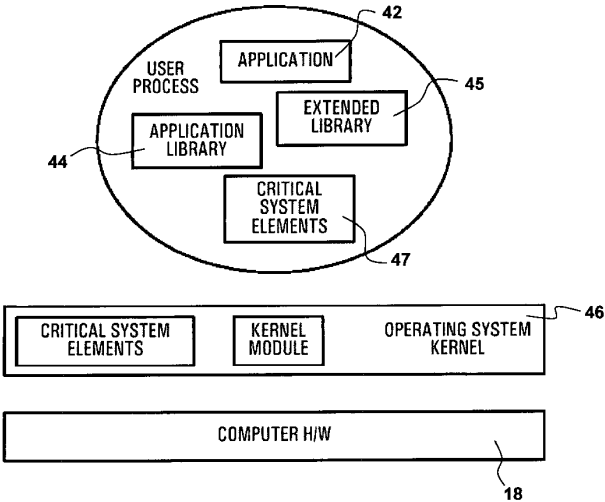
(60) Provisional application No. 60/504,213, filed on Sep.
22, 2003.

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

(51) **Int. Cl.**
G06F 9/22 (2006.01)
(52) **U.S. Cl.** **719/310; 719/319**
(58) **Field of Classification Search** **719/310,**
719/319
See application file for complete search history.

(56) **References Cited**
U.S. PATENT DOCUMENTS
5,481,706 A * 1/1996 Peek 710/200
6,212,574 B1 * 4/2001 O’Rourke et al. 719/321
6,260,075 B1 * 7/2001 Cabrero et al. 719/310

18 Claims, 7 Drawing Sheets



U.S. Patent

Aug. 24, 2010

Sheet 1 of 7

US 7,784,058 B2

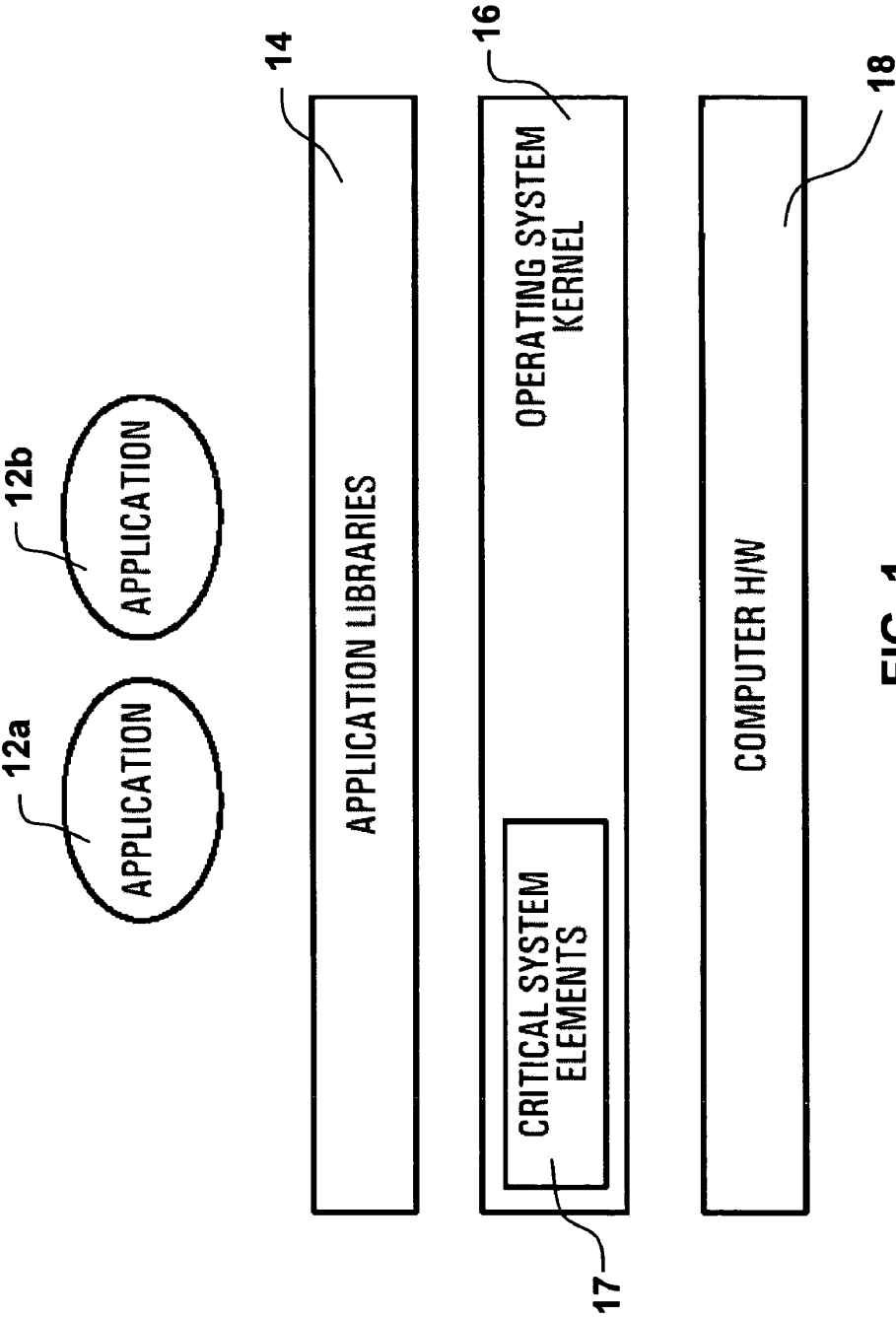


FIG. 1
Prior Art

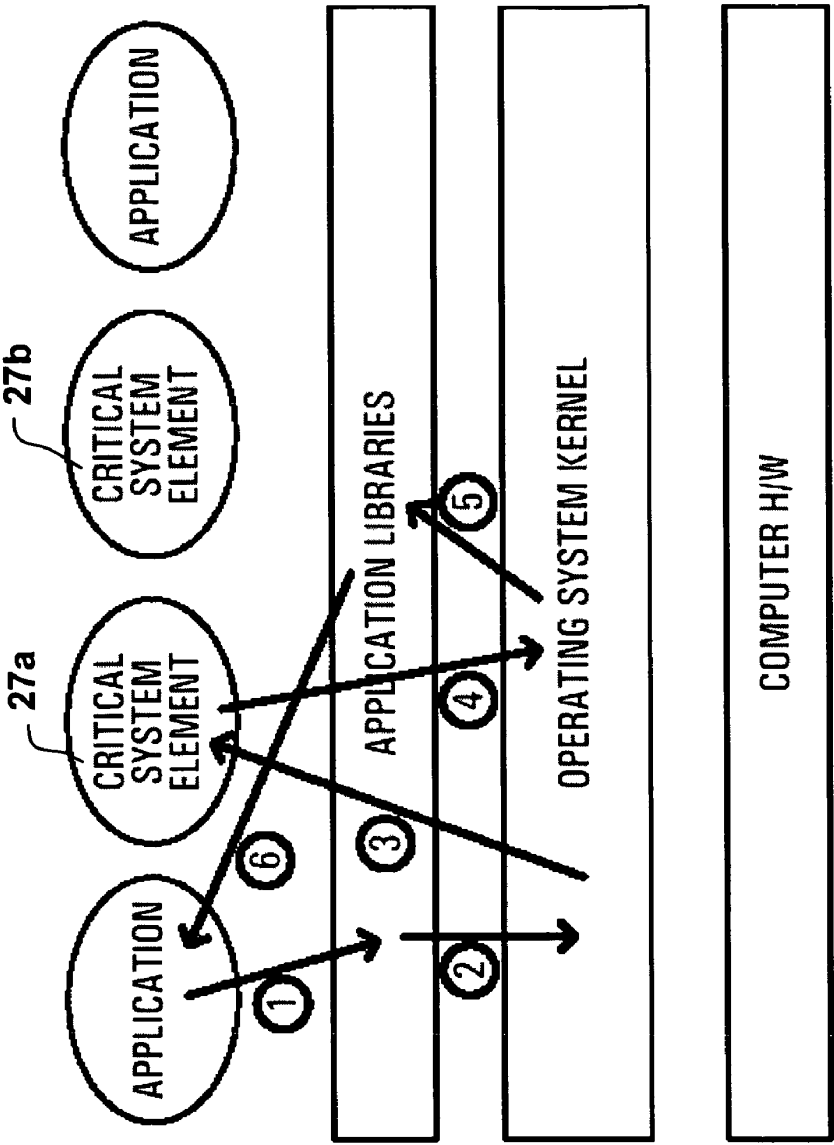


FIG. 2a
Prior Art

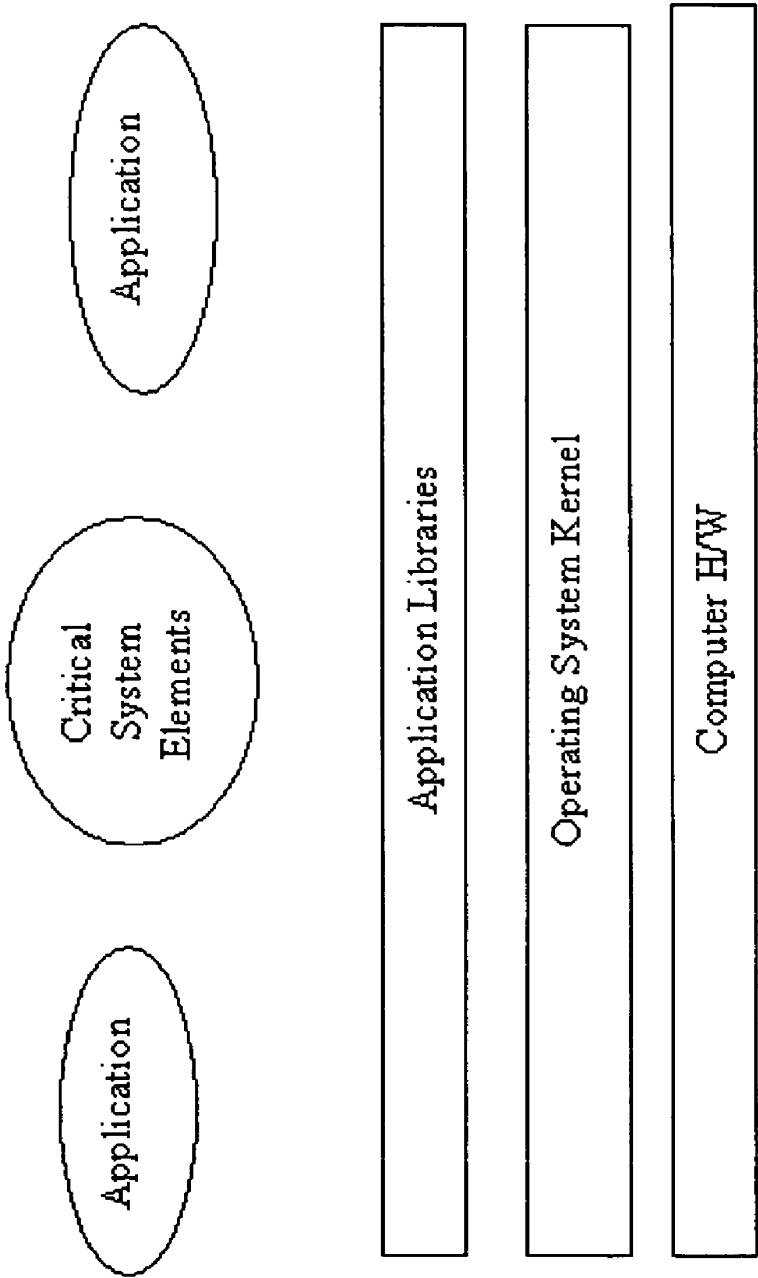


FIG. 2b
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 4 of 7

US 7,784,058 B2

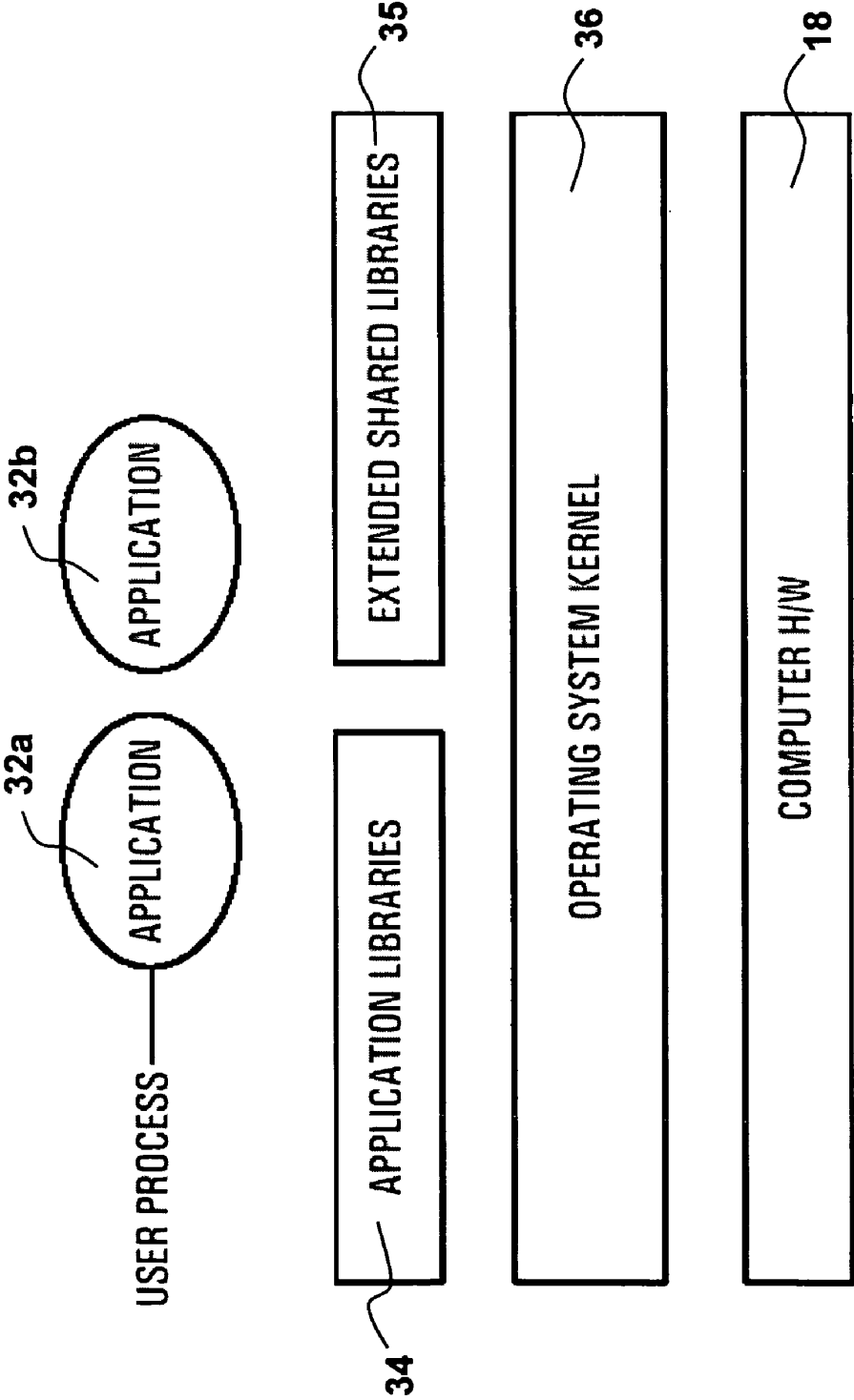


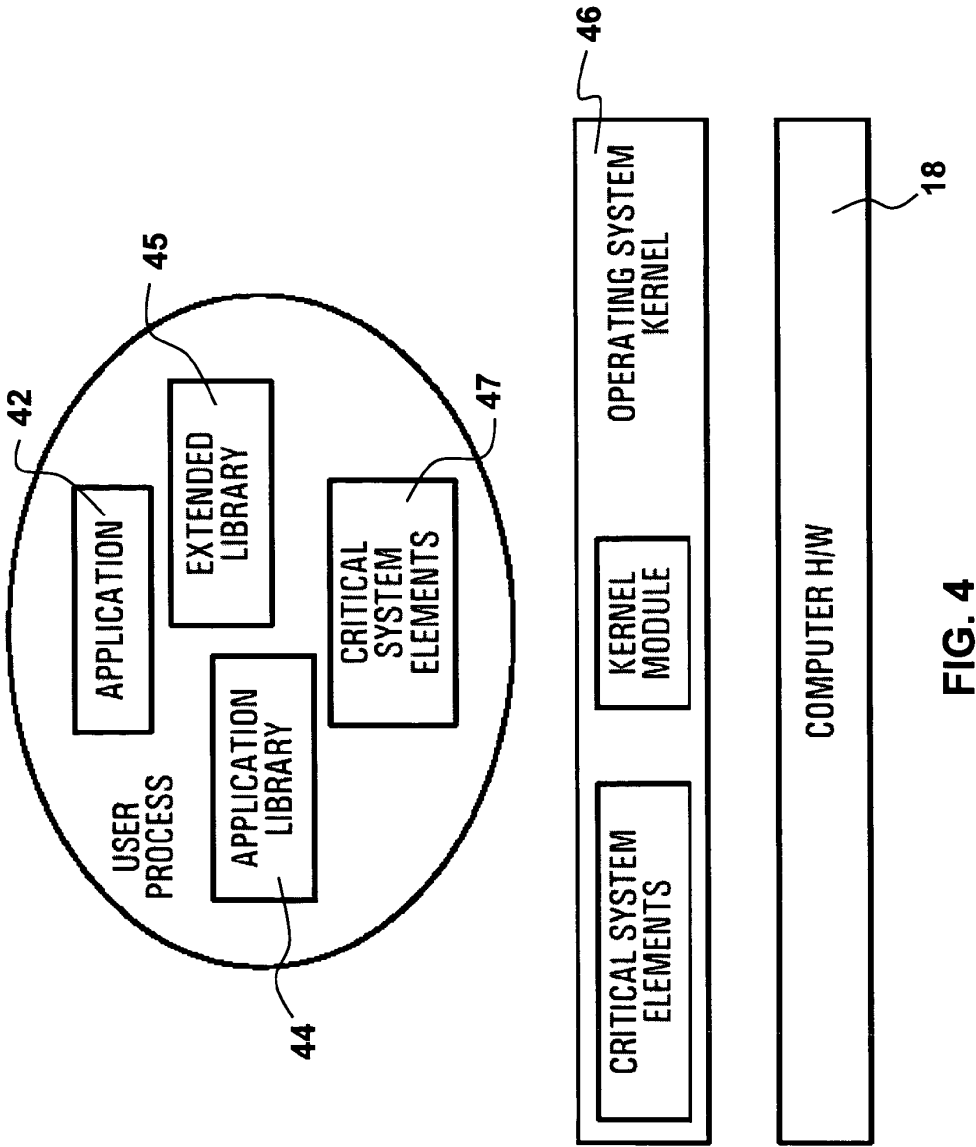
FIG. 3

U.S. Patent

Aug. 24, 2010

Sheet 5 of 7

US 7,784,058 B2



U.S. Patent

Aug. 24, 2010

Sheet 6 of 7

US 7,784,058 B2

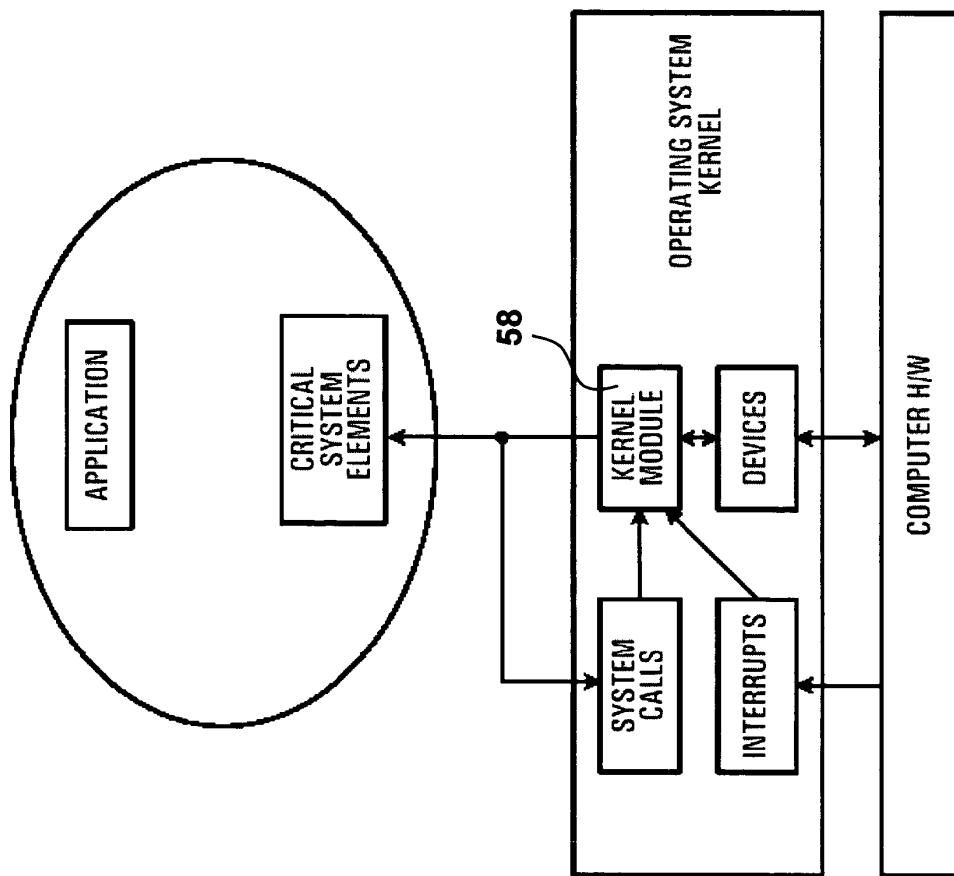


FIG. 5

U.S. Patent

Aug. 24, 2010

Sheet 7 of 7

US 7,784,058 B2

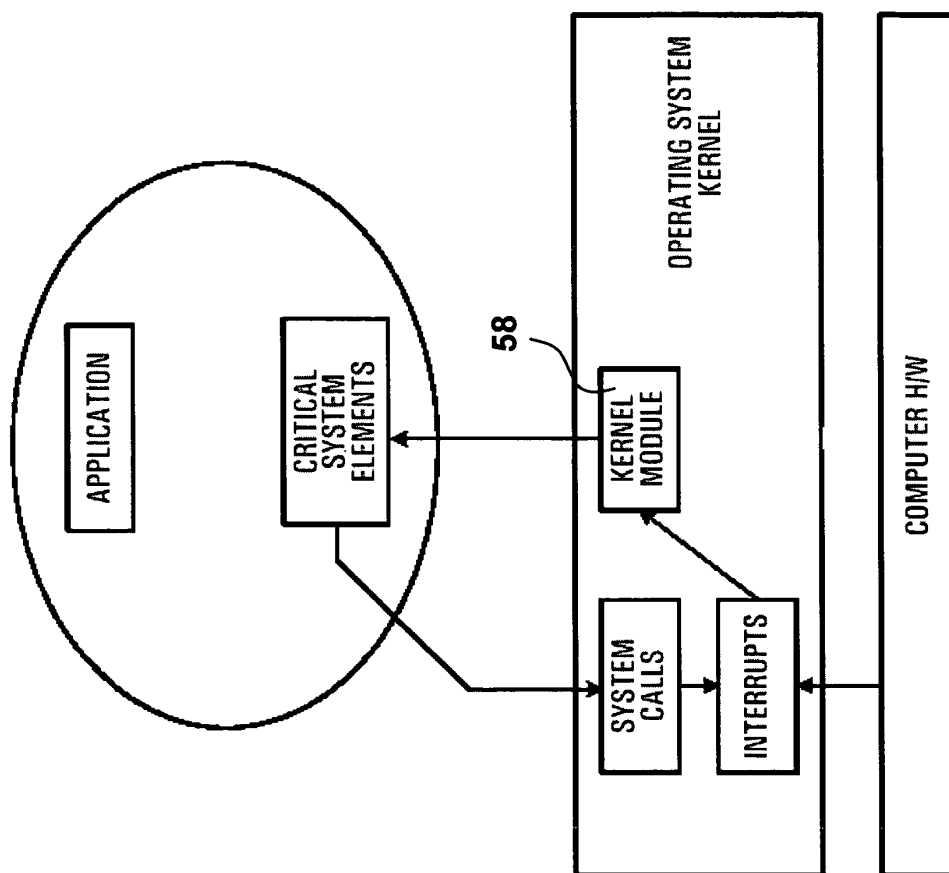


FIG. 6

US 7,784,058 B2

1

**COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled “User Mode Critical System Element as Shared Libs”, which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

2

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.

2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

US 7,784,058 B2

3

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to inter-cept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

4

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. 1 is an architectural view of the traditional monolithic prior art operating system;

US 7,784,058 B2

5

FIG. 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. 3 is an architectural view of an embodiment of the invention;

FIG. 4 is a functional view showing how critical system elements exist in the same context as an application;

FIG. 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. 6 shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TCP/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP

6

stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality

2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided

2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b

US 7,784,058 B2

7

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12a**, **12b** makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12a**, **12b** in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2a** shows a system architecture where critical system elements **27a**, **27b** execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2a** is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2a** as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2b** is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2a** and FIG. **2b** is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2b** is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the

8

case with a multi-server architecture depicted in FIG. **2a**. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32a**, **32b**, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45,47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

US 7,784,058 B2

9

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. 4. FIG. 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

10

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:
 - a) a processor;
 - b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,
 - c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

US 7,784,058 B2

11

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and
iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.
2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.
3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.
4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.
5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.
6. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.
7. A computing system according to claim 6 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.
8. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

12

9. A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.
10. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.
11. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.
12. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.
13. A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.
14. A computing system according to claim 13 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.
15. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.
16. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.
17. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.
18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.

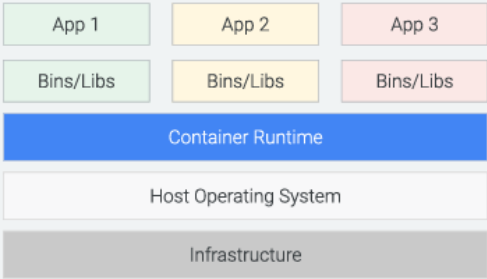
* * * * *

U.S. Patent No. 7,784,058 (“’058 Patent”)

Accused Instrumentalities: Google’s “Migrate to Containers,” and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

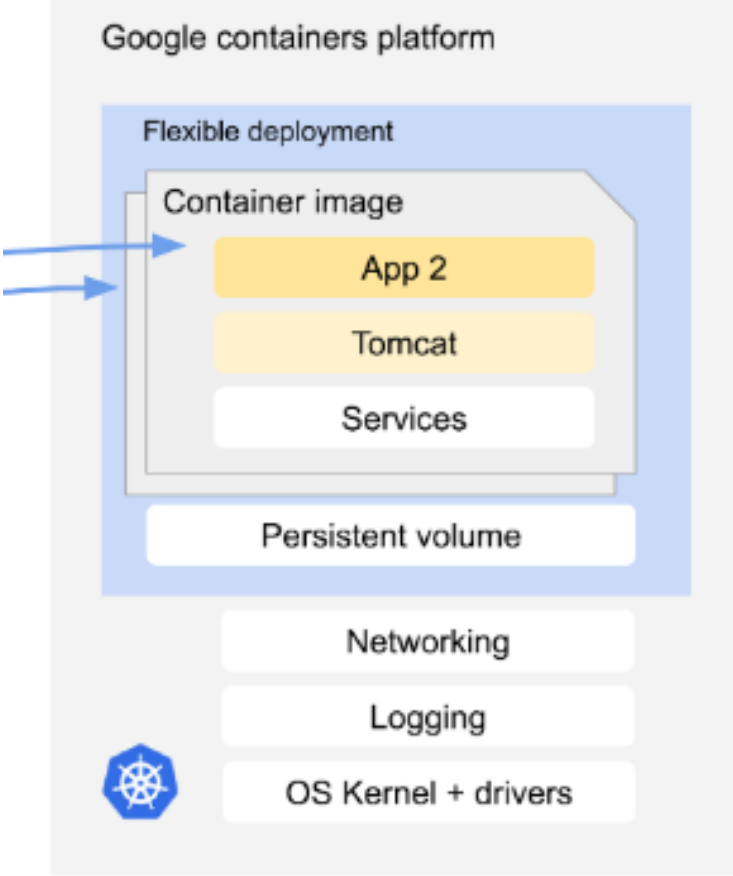
Claim 1	Accused Instrumentalities
<p>[1pre] 1. A computing system for executing a plurality of software applications comprising:</p>	<p>To the extent the preamble is limiting, each Accused Instrumentality comprises or constitutes a computing system for executing a plurality of software applications as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>Use Migrate to Containers to modernize traditional applications away from virtual machine (VM) instances and into native containers that run on Google Kubernetes Engine (GKE), GKE Enterprise clusters, or Cloud Run platform. You can migrate workloads from VMs that run on VMware or Compute Engine, giving you the flexibility to containerize your existing workloads with ease. Migrate to Containers supports modernization of IBM WebSphere, JBoss, Apache, Tomcat, WordPress, Windows IIS applications, as well as containerisation of Linux-based applications.</p> <p>https://cloud.google.com/migrate/containers/docs/getting-started.</p> <p>A container is a way of packaging a given application’s code and dependencies so that the application will run easily in any computing environment. This solves the common problem of</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="653 315 1220 743"><p>The diagram illustrates a container architecture stack. At the top, three application boxes labeled 'App 1' (green), 'App 2' (yellow), and 'App 3' (pink) are shown. Below each app is a corresponding 'Bins/Libs' box in the same color. These three pairs are stacked vertically. Below the 'Bins/Libs' boxes is a blue box labeled 'Container Runtime'. Underneath that is a light gray box labeled 'Host Operating System'. At the bottom is a dark gray box labeled 'Infrastructure'. The entire stack is enclosed in a light gray box with the label 'Containers' centered below it.</p></div> <p data-bbox="653 769 1598 802">https://services.google.com/fh/files/misc/why_container_security_matters.pdf</p> <p data-bbox="678 841 1182 1247">Containers can run virtually anywhere, greatly easing development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or on physical servers; on a developer's machine or in data centers on-premises; and of course, in the public cloud.</p>

Claim 1	Accused Instrumentalities
	<p>Containers are lightweight packages of your application code together with dependencies such as specific versions of programming language runtimes and libraries required to run your software services.</p> <p>https://cloud.google.com/learn/what-are-containers</p>
[1a] a) a processor;	<p>Each Accused Instrumentality comprises a processor.</p> <p><i>See, e.g.:</i></p> <p>Containers virtualize CPU, memory, storage, and network resources at the operating system level, providing developers with a view of the OS logically isolated from other applications.</p> <p>https://cloud.google.com/learn/what-are-containers</p> <ul style="list-style-type: none">• Higher utilization and density, leveraging automatic bin-packing and auto-scaling capabilities, Kubernetes places containers optimally in nodes based on required resources while scaling as needed, without impairing availability. In addition, unlike VMs, all containers on a single node share one copy of the operating system and don't each require their own OS image and vCPU, resulting in a much smaller memory footprint and CPU needs. This means more workloads running on fewer compute resources.

Claim 1	Accused Instrumentalities
	<p>https://cloud.google.com/blog/products/containers-kubernetes/how-migrate-for-anthos-improves-vm-to-container-migration</p> <p>Containers use specific features of the Linux kernel that “trick” individual applications into thinking they’re in their own unique environment, even though multiple applications share the same host kernel. (If you’re not familiar with the Linux kernel, it’s a part of the operating system that communicates between processes--requests that do user tasks like opening a file, running a program-- and the hardware. It manages resources like memory and CPU to meet these requests).</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf</p>
[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,	<p>Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.</p> <p><i>See, e.g.:</i></p> <ul style="list-style-type: none">• Containers are much more lightweight than VMs• Containers virtualize at the OS level while VMs virtualize at the hardware level• Containers share the OS kernel and use a fraction of the memory VMs require <p>https://cloud.google.com/learn/what-are-containers</p>

Claim 1	Accused Instrumentalities
	<p>Kernel mode</p> <p>Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p>https://www.techtarget.com/searchdatacenter/definition/kernel</p>

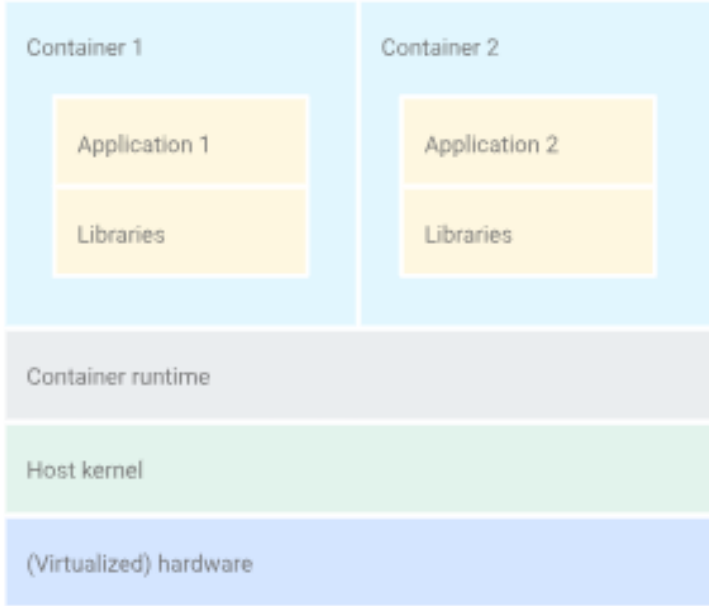
Claim 1	Accused Instrumentalities
	 <p>The diagram illustrates the Google containers platform architecture. At the top is the 'Google containers platform' box. Inside it is the 'Flexible deployment' box. Within 'Flexible deployment' is the 'Container image' box, which contains three stacked components: 'App 2' (yellow), 'Tomcat' (yellow), and 'Services' (white). Below the 'Container image' box is the 'Persistent volume' box. Below the 'Flexible deployment' box are three stacked boxes: 'Networking', 'Logging', and 'OS Kernel + drivers'. A blue Kubernetes logo is positioned to the left of the 'OS Kernel + drivers' box. Two blue arrows point from the left towards the 'Container image' box.</p> <p>https://cloud.google.com/blog/products/application-modernization/shift-your-apps-to-container-based-workloads-on-the-command-line</p>

Claim 1	Accused Instrumentalities
	<p>The migration prerequisites are dependent on your specific migration environment. Confirm that your workloads' OS and source platform are compatible for migration by reviewing the prerequisites for your specific migration environment:</p> <p>https://cloud.google.com/migrate/containers/docs/setting-up-overview</p> <p>Containers use specific features of the Linux kernel that "trick" individual applications into thinking they're in their own unique environment, even though multiple applications share the same host kernel. (If you're not familiar with the Linux kernel, it's a part of the operating system that communicates between processes--requests that do user tasks like opening a file, running a program-- and the hardware. It manages resources like memory and CPU to meet these requests).</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf</p>
[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and	<p>Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.</p> <p><i>See, e.g.:</i></p>

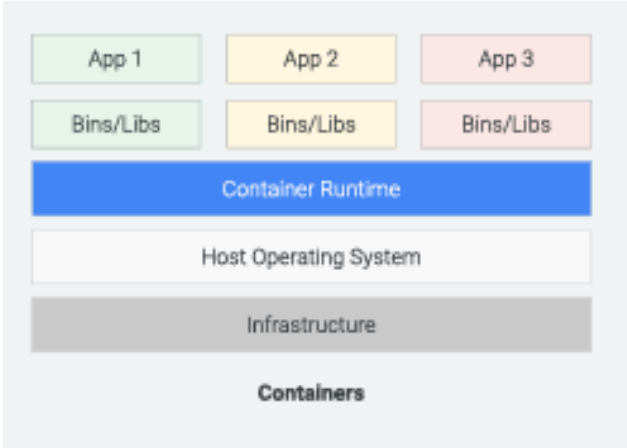
Claim 1	Accused Instrumentalities
	<p data-bbox="680 337 1220 748">A “container image” is your application and its dependencies, and uses a “base image” as the basis for the container image</p> <p data-bbox="680 818 1829 1203">The container image specifies the container’s file system. For example, if you’re running a Node.js application, the container image would contain your app, Node.js, and other dependencies like Linux system libraries (except the kernel). A container image usually extends a base operating system image, or base image. This base image is the basis of your container, so you’ll want to ensure that it’s properly patched and free from known vulnerabilities.</p> <p data-bbox="653 1252 1598 1284">https://services.google.com/fh/files/misc/why_container_security_matters.pdf</p>

Claim 1	Accused Instrumentalities
	<p>A base image is the starting point for most container-based development workflows. Developers start with a base image and layer on top of it the necessary libraries, binaries, and configuration files used to run their application.</p> <p>Many base images are basic or minimal Linux distributions: Debian, Ubuntu, Red Hat Enterprise Linux (RHEL), Rocky Linux, or Alpine. Developers can consume these images directly from Docker Hub or other sources. There are official providers along with a wide variety of other downstream repackagers that layer software to meet customer needs.</p> <p>Google maintains base images for building its own applications. These images are built from the same source that Docker Hub uses. Therefore, they match the images you would get from Docker Hub.</p> <p>https://cloud.google.com/software-supply-chain-security/docs/base-images</p> <p>The preconfigured base images provided by Cloud Workstations contain only a minimal environment with IDE, basic Linux terminal and language tools and a <code>sshd</code> server. To expedite the environment setup of specific development use cases, you can create custom container images that extend these base images to pre-install tools and dependencies and that run automation scripts.</p> <p>For custom container images, we recommend setting up a pipeline to automatically rebuild these images when the Cloud Workstations base image is updated, in addition to running a container scanning tool such as Artifact Analysis to inspect any additional dependencies you added. You're responsible for maintaining and updating custom packages and dependencies added to custom images.</p> <p>https://cloud.google.com/workstations/docs/customize-container-images</p> <p>A container is a way of packaging a given application's code and dependencies so that the application will run easily in any computing environment. This solves the common problem of</p>


Claim 1	Accused Instrumentalities
	<p>Containers solve the portability problem by isolating the application and its dependencies so they can be moved seamlessly between machines. A process running in a container lives isolated from the underlying environment. You control what it can see and what resources it can access. This helps you use resources more efficiently and not worry about the underlying infrastructure.</p> <p>One of the primary reasons to adopt containers is for your applications to be decoupled from the underlying environment and support higher resource utilization by “bin packing” multiple workloads onto each server. As such, the architecture of containers means that they’re deployed with multiple containers sharing the same kernel.</p> <p>The core components of the Linux kernel that are used for containers are cgroups — control groups, which define the resources like CPU and memory which are available to a given process — and namespaces, which are a way of separating processes by restricting what each process can see, so that system resources “appear” isolated to the process.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf</p>

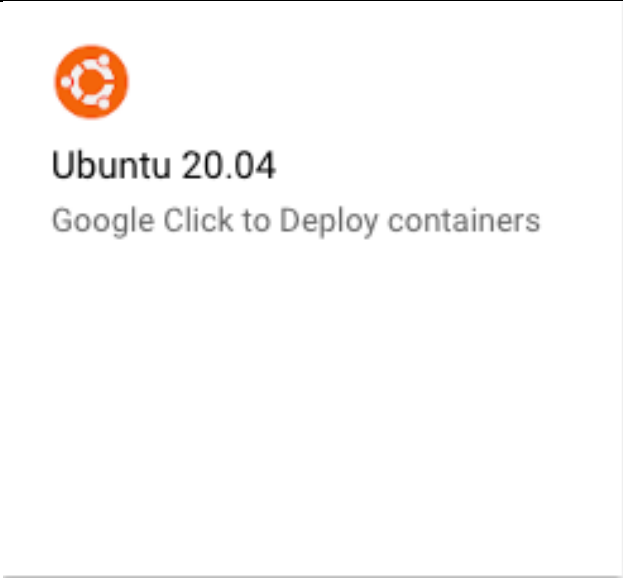
Claim 1	Accused Instrumentalities
	 <p>The diagram illustrates a container architecture stack. At the top, two containers are shown side-by-side: 'Container 1' and 'Container 2'. Inside 'Container 1' are 'Application 1' and 'Libraries'. Inside 'Container 2' are 'Application 2' and 'Libraries'. Below the containers is a grey bar labeled 'Container runtime'. Below that is a green bar labeled 'Host kernel'. At the bottom is a blue bar labeled '(Virtualized) hardware'.</p> <p>https://cloud.google.com/architecture/best-practices-for-operating-containers</p> <p>For example, Migrate to Containers automatically generates a container image, a Dockerfile for day-2 image updates and application revisions, Kubernetes deployment YAMLs and (where relevant) a persistent data volume onto which the application data files and persistent state are copied. This automated, intelligent extraction is</p> <p>https://cloud.google.com/blog/products/containers-kubernetes/how-migrate-for-anthos-improves-vm-to-container-migration</p>

Claim 1	Accused Instrumentalities
	<p>Containers are lightweight packages of your application code together with dependencies such as specific versions of programming language runtimes and libraries required to run your software services.</p> <p>https://cloud.google.com/learn/what-are-containers</p>
<p>[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,</p>	<p>In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.</p> <p>For example, a Docker base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and dependencies, including system libraries or shared library critical system elements (SLCSEs). When the container runs the image, it creates a runtime instance of that container image.</p> <p><i>See, e.g.:</i></p> <p>Many base images are basic or minimal Linux distributions: Debian, Ubuntu, Red Hat Enterprise Linux (RHEL), Rocky Linux, or Alpine. Developers can consume these images directly from Docker Hub or other sources. There are official providers along with a wide variety of other downstream repackagers that layer software to meet customer needs.</p> <p>https://cloud.google.com/software-supply-chain-security/docs/base-images</p>

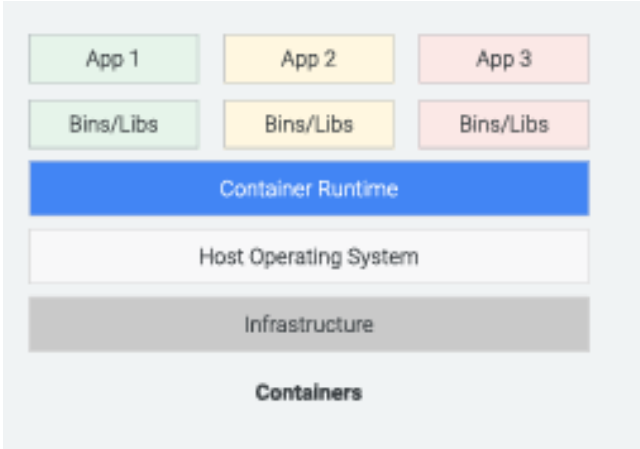
Claim 1	Accused Instrumentalities
	<p data-bbox="680 350 1793 509">A container is a way of packaging a given application's code and dependencies so that the application will run easily in any computing environment. This solves the common problem of</p> <p data-bbox="672 558 1010 821">A "container image" is your application and its dependencies, and uses a "base image" as the basis for the container image</p>  <p>The diagram illustrates the container architecture stack. At the top, three application boxes labeled 'App 1' (green), 'App 2' (yellow), and 'App 3' (pink) are shown. Below each app is a corresponding 'Bins/Libs' box in the same color. These are all contained within a blue 'Container Runtime' box. Below the runtime is a white 'Host Operating System' box, and at the bottom is a grey 'Infrastructure' box. The entire stack is labeled 'Containers' at the bottom.</p>

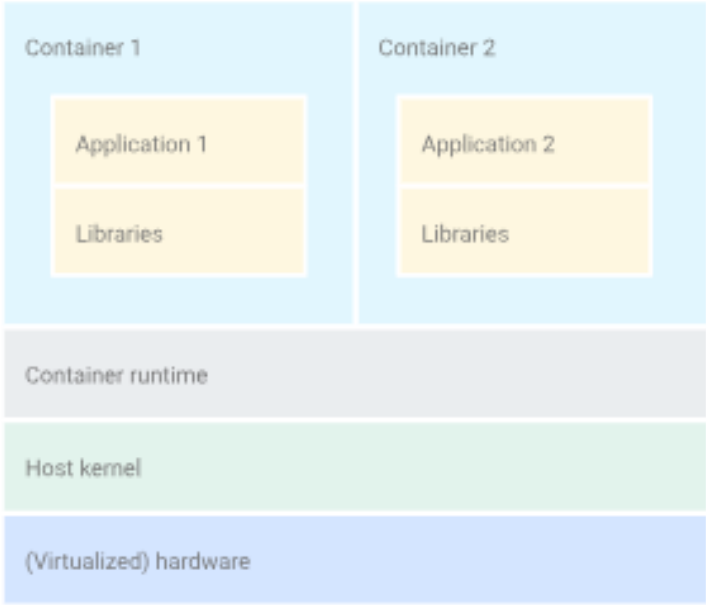
Claim 1	Accused Instrumentalities																								
	<p>The container image specifies the container's file system. For example, if you're running a Node.js application, the container image would contain your app, Node.js, and other dependencies like Linux system libraries (except the kernel). A container image usually extends a base operating system image, or base image. This base image is the basis of your container, so you'll want to ensure that it's properly patched and free from known vulnerabilities.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf</p> <table><tr><th>OS</th><th>Repository path</th><th>Google Cloud Marketplace listing</th></tr><tr><td>Debian 10 "Buster"</td><td><code>marketplace.gcr.io/google/debian10</code></td><td>Google Cloud Marketplace</td></tr><tr><td>Debian 11 "Bullseye"</td><td><code>marketplace.gcr.io/google/debian11</code></td><td>Google Cloud Marketplace</td></tr><tr><td>Debian 12 "Bookworm"</td><td><code>marketplace.gcr.io/google/debian12</code></td><td>Google Cloud Marketplace</td></tr><tr><td>Rocky Linux 8</td><td><code>marketplace.gcr.io/google/rockylinux8</code></td><td>Google Cloud Marketplace</td></tr><tr><td>Rocky Linux 9</td><td><code>marketplace.gcr.io/google/rockylinux9</code></td><td>Google Cloud Marketplace</td></tr><tr><td>Ubuntu 20.04</td><td><code>marketplace.gcr.io/google/ubuntu2004</code></td><td>Google Cloud Marketplace</td></tr><tr><td>Ubuntu 22.04</td><td><code>marketplace.gcr.io/google/ubuntu2204</code></td><td>Google Cloud Marketplace</td></tr></table> <p>https://cloud.google.com/software-supply-chain-security/docs/base-images</p>	OS	Repository path	Google Cloud Marketplace listing	Debian 10 "Buster"	<code>marketplace.gcr.io/google/debian10</code>	Google Cloud Marketplace	Debian 11 "Bullseye"	<code>marketplace.gcr.io/google/debian11</code>	Google Cloud Marketplace	Debian 12 "Bookworm"	<code>marketplace.gcr.io/google/debian12</code>	Google Cloud Marketplace	Rocky Linux 8	<code>marketplace.gcr.io/google/rockylinux8</code>	Google Cloud Marketplace	Rocky Linux 9	<code>marketplace.gcr.io/google/rockylinux9</code>	Google Cloud Marketplace	Ubuntu 20.04	<code>marketplace.gcr.io/google/ubuntu2004</code>	Google Cloud Marketplace	Ubuntu 22.04	<code>marketplace.gcr.io/google/ubuntu2204</code>	Google Cloud Marketplace
OS	Repository path	Google Cloud Marketplace listing																							
Debian 10 "Buster"	<code>marketplace.gcr.io/google/debian10</code>	Google Cloud Marketplace																							
Debian 11 "Bullseye"	<code>marketplace.gcr.io/google/debian11</code>	Google Cloud Marketplace																							
Debian 12 "Bookworm"	<code>marketplace.gcr.io/google/debian12</code>	Google Cloud Marketplace																							
Rocky Linux 8	<code>marketplace.gcr.io/google/rockylinux8</code>	Google Cloud Marketplace																							
Rocky Linux 9	<code>marketplace.gcr.io/google/rockylinux9</code>	Google Cloud Marketplace																							
Ubuntu 20.04	<code>marketplace.gcr.io/google/ubuntu2004</code>	Google Cloud Marketplace																							
Ubuntu 22.04	<code>marketplace.gcr.io/google/ubuntu2204</code>	Google Cloud Marketplace																							

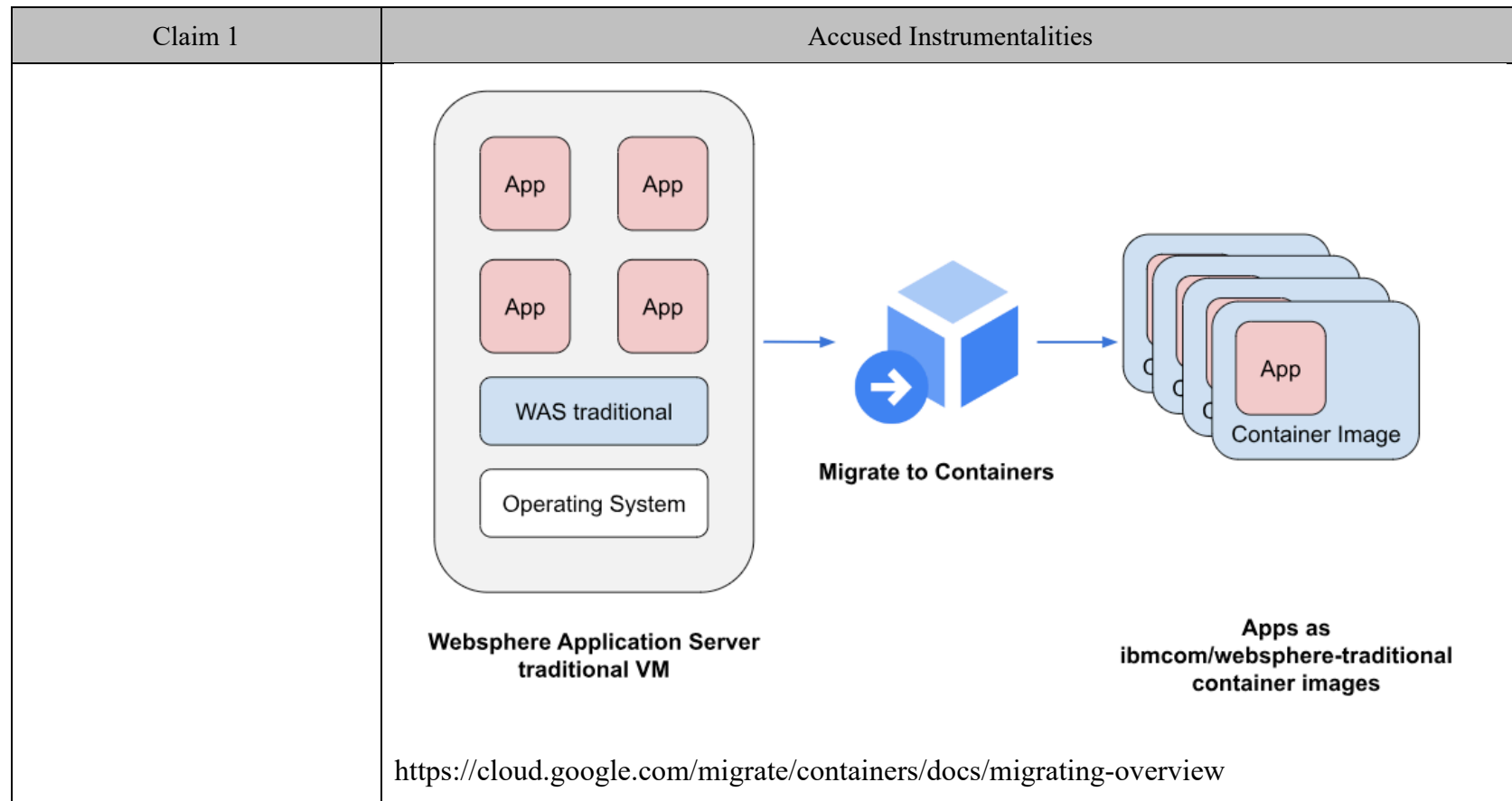
Claim 1	Accused Instrumentalities
	 <p data-bbox="709 459 1020 500">Debian 10 "Buster"</p> <p data-bbox="709 516 1199 557">Google Click to Deploy containers</p> <p data-bbox="709 597 947 638">Open source OS</p>

Claim 1	Accused Instrumentalities
	 <p data-bbox="653 922 1675 951">https://console.cloud.google.com/marketplace/browse?filter=solution-type:container</p>
<p>[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running</p>	<p>In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.</p> <p>When a Docker image is used to create a container, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications running in their respective containers. The docker image includes essential system files, libraries, and dependencies required to run the software application within the container. The Docker</p>

Claim 1	Accused Instrumentalities
<p>under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and</p>	<p>containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it operates in isolation and runs its own instance of the software application without sharing resources or critical system elements with other containers. This ensures that each container has its own isolated context. Docker containers can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same or a different function.</p> <p><i>See, e.g.:</i></p> <p>A container is a way of packaging a given application's code and dependencies so that the application will run easily in any computing environment. This solves the common problem of</p> <p>Containers solve the portability problem by isolating the application and its dependencies so they can be moved seamlessly between machines. A process running in a container lives isolated from the underlying environment. You control what it can see and what resources it can access. This helps you use resources more efficiently and not worry about the underlying infrastructure.</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="688 347 1444 602">The container image specifies the container's file system. For example, if you're running a Node.js application, the container image would contain your app, Node.js, and other dependencies like Linux system libraries (except the kernel). A container image usually extends a base operating system image, or base image. This base image is the basis of your container, so you'll want to ensure that it's properly patched and free from known vulnerabilities.</p>  <p data-bbox="653 1114 1598 1146">https://services.google.com/fh/files/misc/why_container_security_matters.pdf</p>

Claim 1	Accused Instrumentalities
	 <p>The diagram illustrates a container architecture stack. At the top, two containers are shown side-by-side: 'Container 1' and 'Container 2'. Inside 'Container 1' are 'Application 1' and 'Libraries'. Inside 'Container 2' are 'Application 2' and 'Libraries'. Below the containers is a grey bar labeled 'Container runtime'. Below that is a green bar labeled 'Host kernel'. At the bottom is a blue bar labeled '(Virtualized) hardware'.</p> <p>https://cloud.google.com/architecture/best-practices-for-operating-containers</p>



Claim 1	Accused Instrumentalities
	<div data-bbox="711 354 1782 997"><div data-bbox="848 362 1052 391">Dockerfile App 1</div><div data-bbox="747 467 982 496">FROM node:19.7.0</div><div data-bbox="747 597 1052 626">ADD src_app1 /src/</div><div data-bbox="747 724 1020 789">RUN cd /src && \ npm install</div><div data-bbox="1436 362 1640 391">Dockerfile App 2</div><div data-bbox="1337 467 1572 496">FROM node:19.7.0</div><div data-bbox="1337 597 1640 626">ADD src_app2 /src/</div><div data-bbox="1337 724 1610 789">RUN cd /src && \ npm install</div><div data-bbox="716 889 1266 997"><div data-bbox="716 889 758 930"></div> Common layers, downloaded only once <div data-bbox="716 954 758 995"></div> Layers unique to each image</div></div> <p data-bbox="653 1081 1646 1110">https://cloud.google.com/architecture/best-practices-for-building-containers</p> <p data-bbox="653 1149 1860 1250">One method of packaging an application into a container is with the use of a Dockerfile. The Dockerfile is similar to a script which instructs the daemon on how to assemble the container image. See the Dockerfile reference documentation for more information.</p>

Claim 1	Accused Instrumentalities
	<p>Using the Dockerfile method to build a container requires direct knowledge about the application in order to assemble the container. The first step to creating a Dockerfile is selecting an image that will be used as the basis of your image. This image should be a parent or base image maintained and published by a trusted source, usually your company.</p> <p>https://codelabs.developers.google.com/developing-containers-with-dockerfiles#2</p>
<p>[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p>	<p>In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p> <p>For example, In Docker, each container operates independently, and a Docker base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When a Docker image is used to create a container in ECS, an instance of the SLCSE is provided to a software application. Therefore, different instances of the SLCSE are provided to different applications for performing either a same or a different function, simultaneously.</p> <p><i>See, e.g.:</i></p> <p>Containers solve the portability problem by isolating the application and its dependencies so they can be moved seamlessly between machines. A process running in a container lives isolated from the underlying environment. You control what it can see and what resources it can access. This helps you use resources more efficiently and not worry about the underlying infrastructure.</p>

Claim 1	Accused Instrumentalities
	<p>The container image specifies the container's file system. For example, if you're running a Node.js application, the container image would contain your app, Node.js, and other dependencies like Linux system libraries (except the kernel). A container image usually extends a base operating system image, or base image. This base image is the basis of your container, so you'll want to ensure that it's properly patched and free from known vulnerabilities.</p> <p>https://services.google.com/fh/files/misc/why_container_security_matters.pdf</p>

Claim 1	Accused Instrumentalities
	<div><div><div data-bbox="848 358 1052 391">Dockerfile App 1</div><div data-bbox="716 431 1190 824"><div data-bbox="716 431 1190 535">FROM node:19.7.0</div><div data-bbox="716 555 1190 672">ADD src_app1 /src/</div><div data-bbox="716 691 1190 824">RUN cd /src && \ npm install</div></div></div><div><div data-bbox="1440 358 1644 391">Dockerfile App 2</div><div data-bbox="1308 431 1782 824"><div data-bbox="1308 431 1782 535">FROM node:19.7.0</div><div data-bbox="1308 555 1782 672">ADD src_app2 /src/</div><div data-bbox="1308 691 1782 824">RUN cd /src && \ npm install</div></div></div><div><div data-bbox="716 886 758 930"></div>Common layers, downloaded only once</div><div><div data-bbox="716 954 758 998"></div>Layers unique to each image</div><div data-bbox="655 1078 1644 1114">https://cloud.google.com/architecture/best-practices-for-building-containers</div></div>

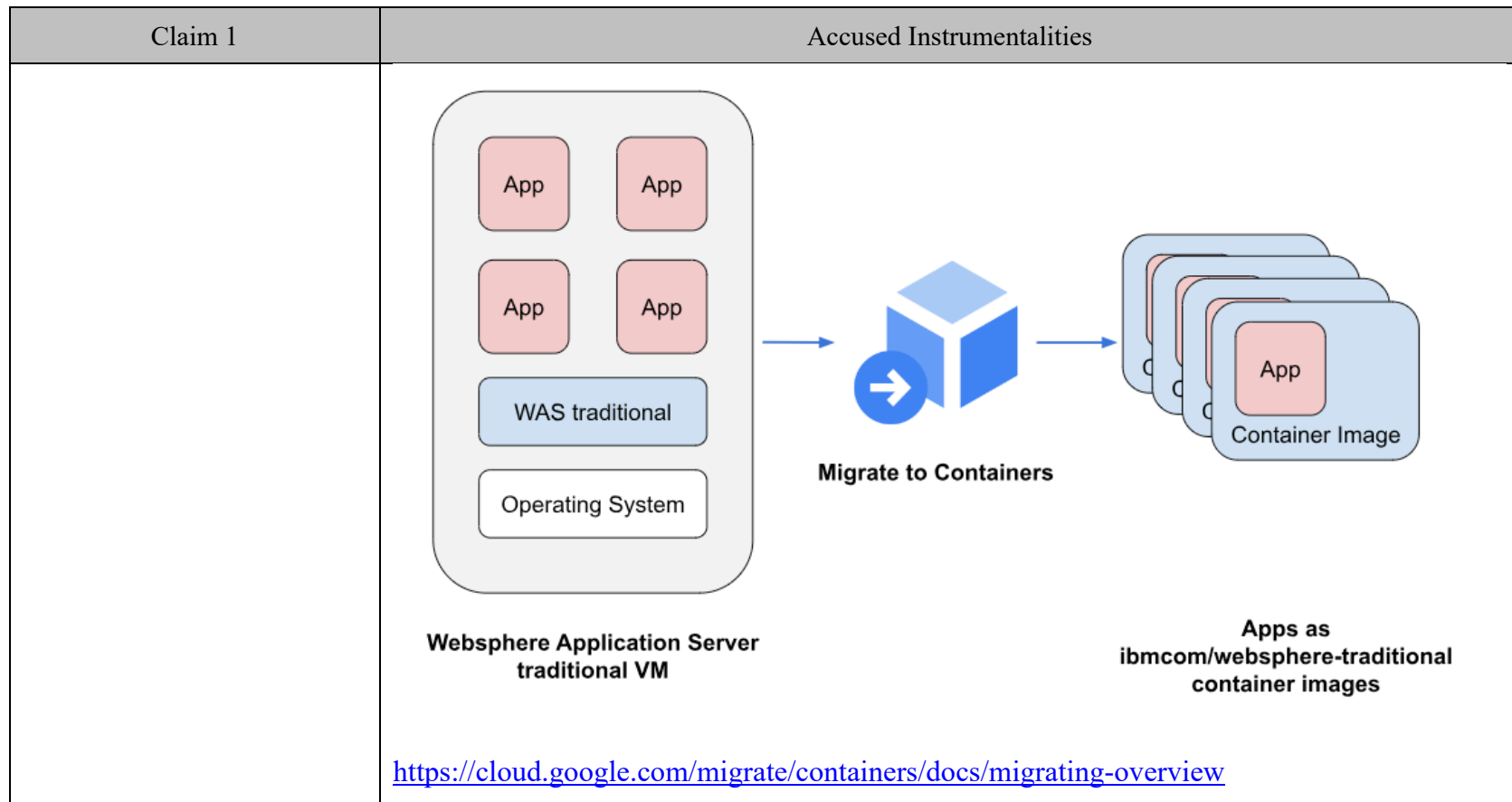


Exhibit 5



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
 United States Patent and Trademark Office
 Address: COMMISSIONER FOR PATENTS
 P.O. Box 1450
 Alexandria, Virginia 22313-1450
 www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
 1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
 P.O. BOX 3791
 ORLANDO, FL 32802-3791

EXAMINER

BAUM, RONALD

ART UNIT

PAPER NUMBER

2439

DATE MAILED: 12/10/2008

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	\$755	\$300	\$0	\$1055	03/10/2009

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE DOES NOT REFLECT A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE IN THIS APPLICATION. IF AN ISSUE FEE HAS PREVIOUSLY BEEN PAID IN THIS APPLICATION (AS SHOWN ABOVE), THE RETURN OF PART B OF THIS FORM WILL BE CONSIDERED A REQUEST TO REAPPLY THE PREVIOUSLY PAID ISSUE FEE TOWARD THE ISSUE FEE NOW DUE.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status above is to be removed, check box 5b on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check box 5a on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and 1/2 the ISSUE FEE shown above.

II. PART B - FEE(S) TRANSMITTAL, or its equivalent, must be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted. If an equivalent of Part B is filed, a request to reapply a previously paid issue fee must be clearly made, and delays in processing may occur due to the difficulty in recognizing the paper as an equivalent of Part B.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: **Mail** **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
 1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
 P.O. BOX 3791
 ORLANDO, FL 32802-3791

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	\$755	\$300	\$0	\$1055	03/10/2009

EXAMINER	ART UNIT	CLASS-SUBCLASS
BAUM, RONALD	2439	713-167000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).

- ☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list

- (1) the names of up to 3 registered patent attorneys or agents OR, alternatively, 1 _____
 (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed. 2 _____
 3 _____

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE

(B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent): ☐ Individual ☐ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:

- ☐ Issue Fee
☐ Publication Fee (No small entity discount permitted)
☐ Advance Order - # of Copies _____

4b. Payment of Fee(s): (Please first reapply any previously paid issue fee shown above)

- ☐ A check is enclosed.
☐ Payment by credit card. Form PTO-2038 is attached.
☐ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number _____ (enclose an extra copy of this form).

5. Change in Entity Status (from status indicated above)

- ☐ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27. ☐ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature _____

Date _____

Typed or printed name _____

Registration No. _____

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
 United States Patent and Trademark Office
 Address: COMMISSIONER FOR PATENTS
 P.O. Box 1450
 Alexandria, Virginia 22313-1450
 www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216
27975	7590	12/10/2008	EXAMINER	
ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A. 1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE P.O. BOX 3791 ORLANDO, FL 32802-3791			BAUM, RONALD	
			ART UNIT	PAPER NUMBER
			2439	
DATE MAILED: 12/10/2008				

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
 (application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 933 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 933 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (<http://pair.uspto.gov>).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at 1-(888)-786-0101 or (571)-272-4200.

Notice of Allowability	Application No.	Applicant(s)	
	10/939,903	ROCHETTE ET AL.	
	Examiner	Art Unit	
	RONALD BAUM	2439	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to 03 September 2008.

2. ☒ The allowed claim(s) is/are 1-34.

3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a) ☐ All b) ☐ Some* c) ☐ None of the:

1. ☐ Certified copies of the priority documents have been received.

2. ☐ Certified copies of the priority documents have been received in Application No. ____.

3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

* Certified copies not received: ____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.

(a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached

1) ☐ hereto or 2) ☐ to Paper No./Mail Date ____.

(b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date ____.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. <input checked="" type="checkbox"/> Notice of References Cited (PTO-892) 2. <input type="checkbox"/> Notice of Draftsperson's Patent Drawing Review (PTO-948) 3. <input type="checkbox"/> Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date ____ 4. <input type="checkbox"/> Examiner's Comment Regarding Requirement for Deposit of Biological Material	5. <input type="checkbox"/> Notice of Informal Patent Application 6. <input type="checkbox"/> Interview Summary (PTO-413), Paper No./Mail Date ____ 7. <input type="checkbox"/> Examiner's Amendment/Comment 8. <input checked="" type="checkbox"/> Examiner's Statement of Reasons for Allowance 9. <input type="checkbox"/> Other ____
---	---

Application/Control Number: 10/939,903
Art Unit: 2439

Page 2

DETAILED ACTION

Examiner's Statement of Reasons for Allowance

1. Claims 1-34 are allowed over prior art.
2. This action is in reply to applicant's correspondence of 03 September 2008.
3. The following is an examiner's statement of reasons for the indication of allowable claimed subject matter.
4. As per claims 1 and 17 generally, prior art of record, Forbes et al, U.S. Patent 6,381,742 B2, fails to anticipate, disclose, teach or suggest alone, or in combination, at the time of the invention, the features as set forth in the claims in this application as allowed, and not necessarily as summarized and/or characterized by the examiner, whether or not as italicized, as discussed and remarked upon in the response of 03 September 2008 to office action of 03 June 2008.

Specifically, (as per claim 1, for example) prior art dealing with various aspects of systems that run multiple applications (often for a single user running unrelated applications scenarios), across independent systems/disparate workstations, utilizing multiple, often different operating systems (i.e., scenarios that require system architectures that require system virtualization with high degrees of both isolation and efficiency (e.g., Hypervisors), alternatives utilizing resource containers and security containers applied to general-purpose, time-shared operating systems exist, such as the Linux-VServer; Soltesz, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., Vol. 41, No. 3. (June 2007), pp. 275-287, entire article, <http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=46675559>), is generally known per se. Nowhere in the prior art is found collectively the *italicized* claim

Application/Control Number: 10/939,903

Page 3

Art Unit: 2439

elements (i.e., the various aspects of applications software not being sharable between the plurality of secure (and isolated) containers of application software, and unique root file systems different from an operating system's root file system, so as to allow for different versions of the same operating system running on the same system/server environment), at the *time of the invention*, serving to patently distinguish the invention from said prior art;

“1. In a system

having *a plurality of servers* with

operating systems that differ,

operating in

disparate computing environments,

wherein each server includes

a processor and

an operating system including

a kernel

a set of associated local system files

compatible with the processor,

a method of providing at least some of the servers in the system with

secure, executable, *applications*

related to a service,

wherein the applications

are executed in

Application/Control Number: 10/939,903
Art Unit: 2439

Page 4

a secure environment,
wherein the *applications each include*
an object *executable by*
at least some of the different operating systems
for performing a task
related to the service,
the method comprising:
storing in memory *accessible to at least some of the servers*
a plurality of secure containers of application software,
each container comprising
one or more of the *executable applications* and
a set of *associated system files*
required to execute the one or more *applications,*
for use with *a local kernel*
residing permanently on
one of the servers;
wherein the set of *associated system files*
are compatible with
a local kernel of
at least *some of the plurality of different operating systems,*
the containers of
application software

Application/Control Number: 10/939,903
Art Unit: 2439

Page 5

excluding a kernel,
wherein some or all of the *associated system files*
within a container stored in memory
are utilized *in place of*
the associated local system files
that remain resident on the server,
wherein *said associated system files*
utilized in place of
the associated *local system files*
are copies or modified copies of
the associated *local system files*
that remain resident on the server, and
wherein the *application software*
cannot be shared between
the plurality of secure containers of application software, and
wherein *each of the containers has*
a unique root file system
that is different from
an operating system's root file system.”

5. Dependent claims 2-16 and 18-34 are allowable by virtue of their dependencies.

Application/Control Number: 10/939,903
Art Unit: 2439

Page 6

Conclusion

6. Any inquiry concerning this communication or earlier communications from examiner should be directed to Ronald Baum, whose telephone number is (571) 272-3861, and whose unofficial Fax number is (571) 273-3861 and unofficial email is Ronald.baum@uspto.gov. The examiner can normally be reached Monday through Thursday from 8:00 AM to 5:30 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kambiz Zand, can be reached at (571) 272-3811. The Fax number for the organization where this application is assigned is **571-273-8300**.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. For more information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Ronald Baum

Patent Examiner

/R. B./

Examiner, Art Unit 2439

/Kambiz Zand/

Supervisory Patent Examiner, Art Unit 2434

Exhibit 6

Notice of Allowability	Application No. 10/946,536	Applicant(s) ROCHETTE ET AL.	
	Examiner SYED RONI	Art Unit 2194	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to 01/15/2010.

2. ☒ The allowed claim(s) is/are 1 - 5 and 7 - 19.

3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
 a) ☐ All b) ☐ Some* c) ☐ None of the:
 1. ☐ Certified copies of the priority documents have been received.
 2. ☐ Certified copies of the priority documents have been received in Application No. _____.
 3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).
 * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
 (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
 1) ☐ hereto or 2) ☐ to Paper No./Mail Date _____.
 (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.
Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. <input type="checkbox"/> Notice of References Cited (PTO-892) 2. <input type="checkbox"/> Notice of Draftperson's Patent Drawing Review (PTO-948) 3. <input type="checkbox"/> Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date _____ 4. <input type="checkbox"/> Examiner's Comment Regarding Requirement for Deposit of Biological Material	5. <input type="checkbox"/> Notice of Informal Patent Application 6. <input checked="" type="checkbox"/> Interview Summary (PTO-413), Paper No./Mail Date <u>20100423</u> . 7. <input checked="" type="checkbox"/> Examiner's Amendment/Comment 8. <input checked="" type="checkbox"/> Examiner's Statement of Reasons for Allowance 9. <input type="checkbox"/> Other _____.
--	--

Application/Control Number: 10/946,536
Art Unit: 2194

Page 2

DETAILED ACTION

EXAMINER'S AMENDMENT

An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview David S. Carus on 04/23/2010.

Claim 1, (Currently Amended) A computing system for executing a plurality of software applications comprising:

a) a processor;

a b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,

b c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, and

Application/Control Number: 10/946,536
 Art Unit: 2194

Page 3

ii) wherein an instance of a an SLCSE provided to at least a first ~~one or more~~ of the plurality of software applications from the shared library is run in a context of said at least first ~~one or more~~ of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second ~~one or more other~~ of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing ~~essentially~~ same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

Claim 3, (Currently Amended) A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing ~~essentially~~ the same function as SLCSEs remain in the operating system kernel.

Claim 6, (Canceled)

Claim 9, (Currently Amended) A computing system according to claim 8 ~~claim 7~~ wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

Application/Control Number: 10/946,536
Art Unit: 2194

Page 4

Claim 19, (Currently Amended) A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs ~~OSLCEs~~.

Claim 20, (Canceled)

Reasons for Allowance

Claims 1 – 5 and 7 – 19 are allowed.

The following is an Examiner's statement of reasons for allowance. None of Elnozahy et al. (hereinafter Elnozahy) (US 7,499,966 B2) and Wong et al. (hereinafter Wong) (US 2004/0216145 A1) on the record discloses "a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, and

ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of

Application/Control Number: 10/946,536

Page 5

Art Unit: 2194

a unique instance of a corresponding critical system element for performing same function.

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously” (Claim 1). Instead, Elnozahy discloses kernel extension device driver that are user space extensions of operating system code to minimize kernel calls by web server and Wong discloses user mode accessible copies of kernel-mode memory to facilitate a device driver to execute in user-mode while the graphics engine remains in kernel mode.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to SYED RONI whose telephone number is (571)270-7806. The examiner can normally be reached on M - F (8:30 am - 5:00 pm).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Hyung Sub Sough (Sam) can be reached on (571) 272 - 6799. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Application/Control Number: 10/946,536

Page 6

Art Unit: 2194

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/SYED RONI/
Examiner, Art Unit 2194

/Hyung S. Sough/
Supervisory Patent Examiner, Art Unit 2194
04/26/10

EXHIBIT D

**UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION**

VIRTAMOVE, CORP.,

Plaintiff,

v.

AMAZON.COM, INC.; AMAZON.COM
SERVICES LLC; AND AMAZON WEB
SERVICES, INC.,

Defendants.

Case No. 7:24-cv-00030-DC-DTG

JURY TRIAL DEMANDED

**FIRST AMENDED COMPLAINT FOR PATENT INFRINGEMENT AGAINST
AMAZON.COM, INC.; AMAZON.COM SERVICES LLC; AND AMAZON WEB
SERVICES, INC.**

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, “Plaintiff” or “VirtaMove”) makes the following allegations against Defendants Amazon.com, Inc.; Amazon.com Services LLC; and Amazon Web Services, Inc. (collectively, “Defendant” or “Amazon”):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant’s unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058 (collectively, the “Asserted Patents”). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of

Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as Appzero Software Corp. (“Appzero”), which was established in 2010.

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repack, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove’s mission is to move and modernize the world’s server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach \$2.1 billion in 2019 and \$4.3 billion in 2022—a compound annual growth rate (“CAGR”) of 30%. *See, e.g.*, <https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022>. Another source reported the application containers market had a market size of \$5.45 billion in 2024 and estimated it to reach \$19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.*, <https://www.mordorintelligence.com/industry-reports/application-container-market>.

6. Defendant Amazon.com, Inc. is a Delaware corporation with a listed registered

agent of Corporation Service Company, 251 Little Falls Drive, Wilmington, Delaware 19808. Amazon has a principal place of business at 410 Terry Ave. North, Seattle, Washington 98109-5210. Amazon may also be served with process via its registered agent Corporation Service Company 300 Deschutes Way SW Ste 208 MC-CSC1, Tumwater, WA, 98501.

7. Defendant Amazon.com Services LLC (formerly “Amazon.com Services Inc.” and referred to herein as “Amazon Services”) is a limited liability company organized under the laws of the state of Delaware, with its principal place of business at 410 Terry Avenue North, Seattle, Washington 98109. Amazon Services is a wholly owned subsidiary of Amazon. Amazon Services is registered to do business in the State of Texas and may be served with process via its registered agent in Texas, Corporation Service Company dba CSC-Lawyers Incorporating Service Company at 211 7th Street, Suite 620, Austin, TX 78701-3218. Amazon Services may also be served via its Delaware registered agent Corporation Service Company, 251 Little Falls Dr., Wilmington, Delaware 19808.

8. Amazon Web Services, Inc. is a Delaware corporation with its principal place of business at 410 Terry Ave. North, Seattle, Washington 98109. Amazon Web Services, Inc. may be served through its registered agent Corporation Service Company, 211 E. 7th Street, Suite 620, Austin, Texas 78701. On information and belief, Amazon Web Services, Inc. is registered to do business in the State of Texas and has been since at least May 3, 2006. On information and belief, Amazon Web Services, LLC is a wholly-owned subsidiary of Amazon.com, Inc.

JURISDICTION AND VENUE

9. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

10. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

11. Venue is proper in this District. For example, Amazon has a regular and established place of business, including, e.g., at Amazon Tech Hub located at 11501 Alterra Parkway, Austin, Texas.

12. Greg O'Connor and other representatives of VirtaMove repeatedly met with Amazon Web Services, Inc. representatives regarding Appzero. Meetings occurred in at least late 2009, 2012-2015, and 2018-2021. During these meetings, VirtaMove demonstrated its Appzero and its technology for legacy app migration and containerization to Amazon Web Services, Inc. Amazon Web Services, Inc. discussed potential partnership with, engagement with, and/or investment in VirtaMove. Amazon Web Services, Inc. would have learned of the Asserted Patents during this time frame, either through disclosure during the meetings, its own due diligence when considering investment, and/or notice through Virtamove's website. *See, e.g.* <https://web.archive.org/web/20230924012816/https://virtamove.com/about-us/product-patents> (listing both Asserted Patents before September 24, 2023); <https://virtamove.com/about-us/product-patents> (listing both Asserted Patents); <https://virtamove.com/blog/virtamove-awarded-a-new-os-upleveling-patent> (announcing '058 Patent); <https://virtamove.com/migration-software/v-migrate> ("VirtaMove's patented software"); <https://virtamove.com/blog/containers->

[are-not-a-windows-server-migration-panacea](#); <https://virtamove.com/blog/virtamove-announces-beta-version-v-migrate-for-linux-container-migrations> (“VirtaMove’s patented software . . . moves them with ease”). Defendants did not end up engaging with, partnering with, or investing in VirtaMove. Instead, Defendants misappropriated Virtamove’s technology and willfully commit the infringement and inducement alleged herein.

COUNT I

INFRINGEMENT OF U.S. PATENT NO. 7,519,814

13. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein. In Count I only, “Defendant” refers to Amazon Web Services, Inc.

14. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 (’814 patent), titled “System for Containerization of Application Sets,” issued on April 14, 2009. A true and correct copy of the ’814 Patent is attached as Exhibit 1.

15. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products (“Accused Products”), such as, e.g., Amazon’s AWS End-of-Support Migration Program (“EMP”), that directly infringe, literally and/or under the doctrine of equivalents, claims of the ’814 patent, for example:

AWS EMP for Windows Server enables application migration from legacy Windows Server systems to supported Windows operating systems. The EMP process includes an assessment of your application and testing requirements, packaging of the application and its dependencies on the legacy operating system based on your requirements, and migration to an AWS environment running a supported Windows Server version. When migrated to the new server environment, the application will run on the new OS as it would on the legacy operating system. The new package redirects application requests and does not include installation of any part of the legacy operating system.

2. The application accesses data stored in a fixed location that is not available on the new OS version: the EMP engine redirects these requests to the appropriate location on the new version of the operating system.

The resulting package includes everything that the application needs to run on a modern operating system, including application files, runtimes, components, and deployment tools. The package does not include the legacy operating system, which means you never run any part of the legacy Windows Server version on the new Windows Server to which the application is upgraded.

The configuration of a package is defined in a series of XML files, which includes:

- Configurations for Registry keys and values (AppRegistry.xml)
- File, Registry, and Network redirections (Redirections.xml)
- File Type Associations (FileAssociations.xml)
- Shortcuts (Shortcuts.xml)
- Executable programs (Programs.xml)
- Environment Variables (EnvironmentVariables.xml)

The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS.

4.1.1 Application and Runtime Isolation

Applications with conflicting requirements or outdated run times can safely run by being isolated from other applications on the server. Isolation enables EMP compatibility packages to include runtimes that conflict with other runtimes on the server, so that they are used only by the packaged application. Examples of conflicting requirements might include use of Java 1.3, .Net 2.0, and msxml.dll.

4.3.1 Application to Operating System Compatibility

EMP software runs legacy applications within a compatibility package on modern, up-to-date operating systems. The EMP compatibility package does not contain any parts of the legacy operating system. Instead, it intercepts the operating system requests and redirects and resolves them against the up-to-date host operating environment.

The EMP compatibility package permits the legacy application to run alongside other applications and/or other versions of the same application. For example, multiple versions of Microsoft Office can run simultaneously on the target operating system, or two incompatible 32-bit applications can run together, isolated from each other by the EMP compatibility package.

4.3.3 Services and Drivers

EMP Compatibility packages support Windows Services out of the box. Drivers aren't captured in an EMP package but can be extracted and deployed locally using the EMP deployment script feature, as long as they are compatible with the target operating system.

https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf.

16. The infringement of the Asserted Patents is also attributable to Defendant.

Defendant and/or users of the Accused Products directs and controls use of the Accused Products

to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

17. Defendant's infringement has been and is willful for the reasons alleged above. Additionally, Defendant knew of VirtaMove, its products, and at least one of the patents long before this suit was filed and at least as early as 2013. For example, on or about October 2013, in connection with the prosecution of U.S. Patent No. 8,806,655 (assigned to Amazon's patent holding entity despite *de facto* invention by Defendant's employees), the examiner cited the '814 Patent against Amazon. On or about June 2020, in connection with the prosecution of U.S. Patent No. 11,061,812 (assigned to Amazon), the examiner cited U.S. Pub. No. 2005/0060722 (which issued as the '814 Patent) against Amazon. Defendant knew, or should have known, that its conduct amounted to infringement of the '814 patent. Accordingly, Defendant is liable for willful infringement.

18. Defendant also knowingly and intentionally induces infringement of claims of the '814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and/or earlier, as set forth above. Despite this knowledge of the '814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '814 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '814 patent, thereby specifically intending for and inducing its customers to infringe the '814 patent through the customers' normal and customary use of the Accused Products.

19. Defendant has also infringed, and continue to infringe, claims of the '814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '814 patent, in violation of 35 U.S.C. § 271(c).

20. The Accused Products satisfy all claim limitations of claims of the '814 patent. A claim chart comparing an independent claim of the '814 patent to a representative Accused Product, is attached as Exhibit 2, which is hereby incorporated by reference in its entirety.

21. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the '814 patent pursuant to 35 U.S.C. § 271.

22. As a result of Defendant's infringement of the '814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. Plaintiff has complied with 35 U.S.C. § 287 through the assertion of method claims, pre-suit marking, and/or actual knowledge by Defendant.

COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

23. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

24. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 ('058 patent), titled "Computing System Having User Mode Critical System Elements as Shared Libraries," issued on August 24, 2010. A true and correct copy of the '058 patent is attached as Exhibit 3.

25. On information and belief, Amazon Web Services, Inc. makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., Amazon's AWS Elastic Container Service ("ECS"), that directly infringe, literally and/or under the doctrine of equivalents, claims of the '058 patent, for example:

Amazon ECS supports using 64-bit ARM applications. You can run your applications on the platform that's powered by [AWS Graviton2](#) processors. It's suitable for a wide variety of workloads. This includes workloads such as application servers, micro-services, high-performance computing, CPU-based machine learning inference, video encoding, electronic design automation, gaming, open-source databases, and in-memory caches.

An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into an Amazon ECS cluster. When you run tasks with Amazon ECS using the EC2 launch type or an Auto Scaling group capacity provider, your tasks are placed on your active container instances.

Note

Tasks using the Fargate launch type are deployed onto infrastructure managed by AWS, so this topic does not apply.

You can use EC2 instances with the following Linux operating systems to run your applications:

- Amazon Linux: This is a general purpose operating system.
- Bottlerocket: Bottlerocket is a Linux based open-source operating system that is purpose built by AWS for running containers on virtual machines or bare metal hosts. The Amazon ECS-optimized Bottlerocket AMI is secure and only includes the minimum number of packages that's required to run containers. This improves resource usage, reduces security attack surface, and helps lower management overhead. For information about the security features and guidance, see [Security Features](#) and [Security Guidance](#) on the GitHub website.

An Amazon ECS container instance specification consists of the following components:

Required

- A Linux distribution running at least version 3.10 of the Linux kernel.
- The Amazon ECS container agent (preferably the latest version). For more information, see [Updating the Amazon ECS container agent](#).

You must architect your applications so that they can run on *containers*. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an *image*. Images are typically built from a Dockerfile. A Dockerfile is a plaintext file that specifies all of the components that are included in the container. After they're built, these images are stored in a *registry* such as Amazon ECR where they can be downloaded from.

Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see [Docker Compose CLI - Amazon ECS](#).

https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf.

26. The infringement of the Asserted Patents is also attributable to Defendant. Amazon.com Services LLC is believed to be responsible for Amazon Prime Video. Amazon.com, Inc. is believed to have an “Amazon Devices & Services” division that is responsible for the Amazon Fire Stick. Amazon Prime Video and the Fire Stick are reported to use ECS sold or offered by Amazon Web Services, Inc.

[Customer Stories](#) / [Media & Entertainment](#) / [Global](#)

2023



Fire TV at Amazon Prime Video Modernizes Its Stack Using Amazon ECS with AWS Fargate

Learn how the Fire TV team at Amazon Prime Video improves engineering productivity and drives innovation by using Amazon ECS with AWS Fargate.

[Overview](#) | [Opportunity](#) | [Solution](#) | [Outcome](#) | [AWS Services Used](#) | [Architecture Diagram](#)

Overview

Subscription-based streaming service [Amazon Prime Video](#) (Prime Video) delivers channels, movies, and television shows to subscribers globally, which they can stream from virtually any device. The Fire TV team at Prime Video, which is responsible for managing the Prime Video app for Fire TV devices, wanted to shift from its shared architecture. This team found its shared architecture hard to scale, and it wanted to build a serverless architecture on Amazon Web Services (AWS) that could scale to millions of active subscribers and simplify deployments and upgrades across devices. Further, the team wanted to free its engineers from the task of provisioning resources manually.

The Fire TV team at Prime Video decided to adopt a backend-for-front-end architecture and rebuilt a dedicated stack for each of the Fire TV streaming experiences. The Fire TV team at Prime Video implemented [Amazon Elastic Container Service](#) (Amazon ECS) to run secure, reliable, scalable, and highly fault-tolerant containers with [AWS Fargate](#), a serverless, pay-as-you-go compute engine for containers. By modernizing its stack, the Fire TV team at Prime Video simplified capacity management and empowered its engineers to focus on improving the viewer experience by accelerating the development of new features and experiences.

<https://aws.amazon.com/solutions/case-studies/amazon-prime-video-ecs-case-study>.

27. Amazon.com, Inc. directs and controls Amazon.com Services LLC and Amazon Web Services, Inc., and they collectively form a joint enterprise marketed as “Amazon” to consumers. On information and belief, Amazon.com, Inc. exercises its direction and control as a parent company to cause Amazon.com Services LLC to offer Prime Video on the Fire Stick, and to use the ECS offerings of Amazon Web Services, Inc., such that Amazon.com, Inc. receives revenue from its subsidiaries and so that Amazon Web Services, Inc. and Amazon.com Services LLC receive corporate support, the use of the “Amazon” trademark, “Amazon” marketing, and other benefits. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

28. Defendant also knowingly and intentionally induces infringement of claims of the ’058 patent in violation of 35 U.S.C. § 271(b) for the reasons alleged above. Additionally, Defendant has had knowledge of the ’058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed. Despite this knowledge of the ’058 patent,

Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '058 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '058 patent, thereby specifically intending for and inducing its customers to infringe the '058 patent through the customers' normal and customary use of the Accused Products.

29. Defendant has also infringed, and continue to infringe, claims of the '058 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '058 patent, in violation of 35 U.S.C. § 271(c).

30. The Accused Products satisfy all claim limitations of claims of the '058 patent. A claim chart comparing an independent claim of the '058 patent to a representative Accused Product, is attached as Exhibit 4, which is hereby incorporated by reference in its entirety.

31. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the '058 patent pursuant to 35 U.S.C. § 271.

32. As a result of Defendant's infringement of the '058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no

event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. Plaintiff has complied with 35 U.S.C. § 287 through pre-suit marking and/or actual knowledge by Defendants.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

- a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;
- b. A judgment in favor of Plaintiff that Defendant has willfully infringed the '814 patent;
- c. A permanent injunction prohibiting Defendant from further acts of infringement of each of the '814 and '058 patents;
- d. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant's infringement of each of the Asserted Patents;
- e. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;
- f. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys' fees against Defendant; and
- g. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury of any issues so triable by right.

Dated: May 3, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)

rmirzaie@raklaw.com

Marc A. Fenster (CA SBN 181067)

mfenster@raklaw.com

Neil A. Rubin (CA SBN 250761)

nrubin@raklaw.com

Amy E. Hayden (CA SBN 287026)

ahayden@raklaw.com

Christian W. Conkle (CA SBN 306374)

cconkle@raklaw.com

Jonathan Ma (CA SBN 312773)

jma@raklaw.com

RUSS AUGUST & KABAT

12424 Wilshire Boulevard, 12th Floor

Los Angeles, CA 90025

Telephone: (310) 826-7474

Qi (Peter) Tong (TX SBN 24119042)

ptong@raklaw.com

RUSS AUGUST & KABAT

4925 Greenville Ave, Suite 200

Dallas, TX 75206

Attorneys for Plaintiff VirtaMove, Corp.

CERTIFICATE OF SERVICE

I certify that on May 3, 2024, a true and correct copy of the foregoing document was electronically filed with the Court and served on all parties of record via the Court's CM/ECF system.

/s/ Reza Mirzaie

Reza Mirzaie

Exhibit 1



US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,519,814 B2**
(45) **Date of Patent:** **Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

WO WO 2006/039181 A 4/2006

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248; 709/214; 719/319

(58) **Field of Classification Search** 713/167; 719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

OTHER PUBLICATIONS

Soltész, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667.*

Primary Examiner—Kambiz Zand

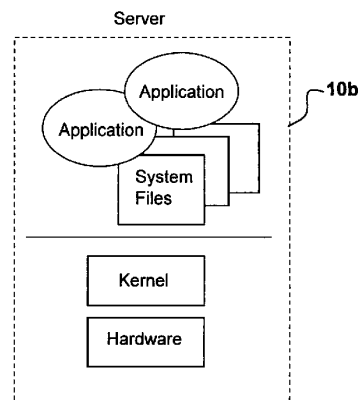
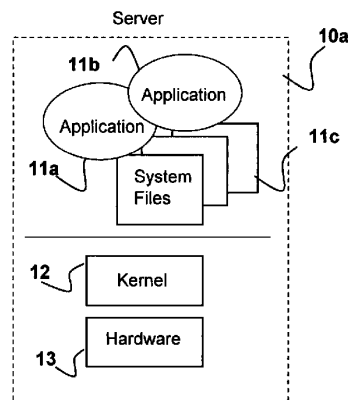
Assistant Examiner—Ronald Baum

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



US 7,519,814 B2

Page 2

U.S. PATENT DOCUMENTS				2002/0004854 A1	1/2002	Hartley	
6,847,970 B2 *	1/2005	Keller et al.	707/100	2002/0174215 A1	11/2002	Schaefer	709/224
7,076,784 B1 *	7/2006	Russell et al.	719/315	2003/0101292 A1	5/2003	Fisher	
7,287,259 B2 *	10/2007	Grier et al.	719/331	* cited by examiner			

U.S. Patent

Apr. 14, 2009

Sheet 1 of 17

US 7,519,814 B2

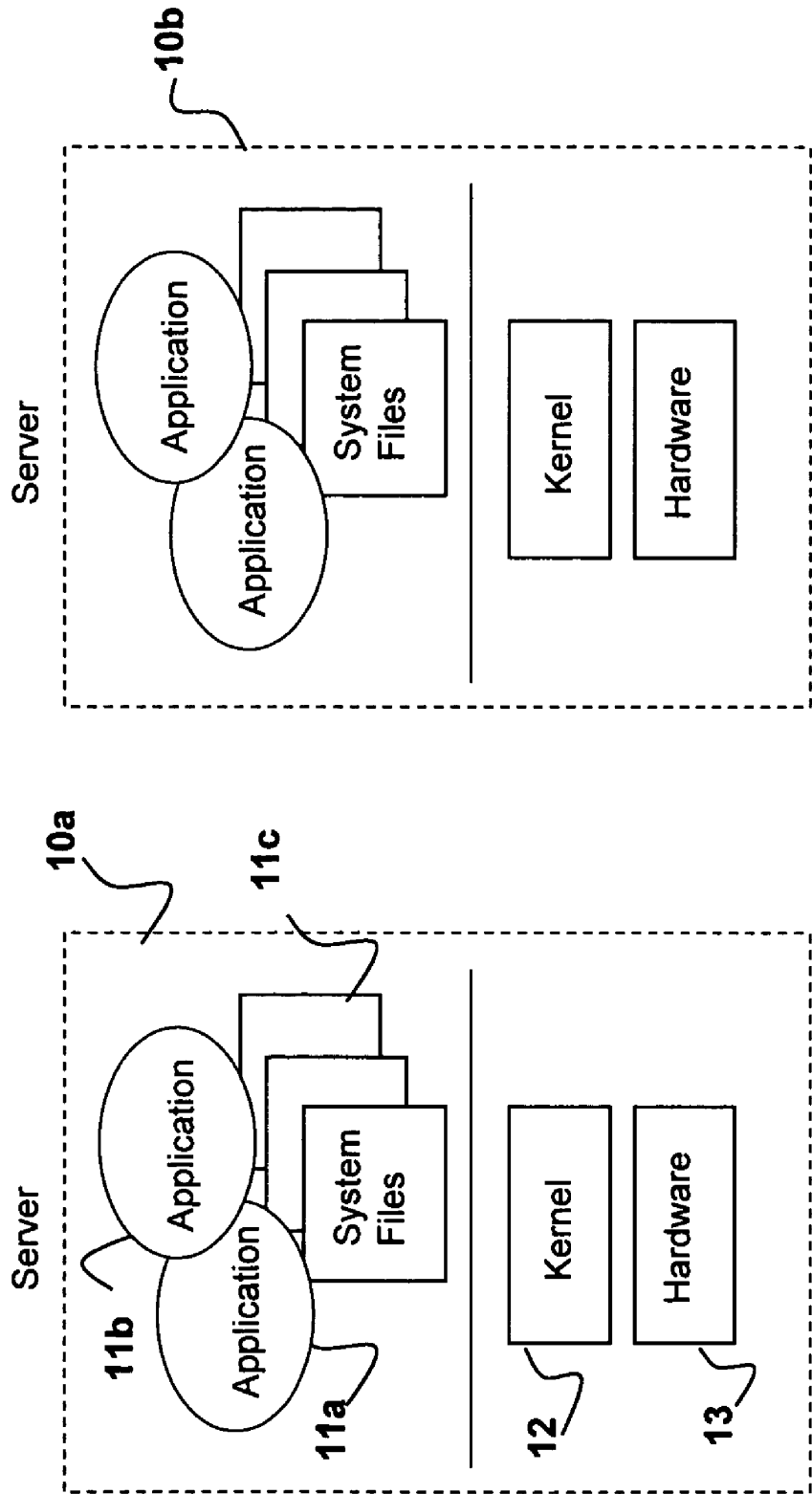


Figure 1

U.S. Patent

Apr. 14, 2009

Sheet 2 of 17

US 7,519,814 B2

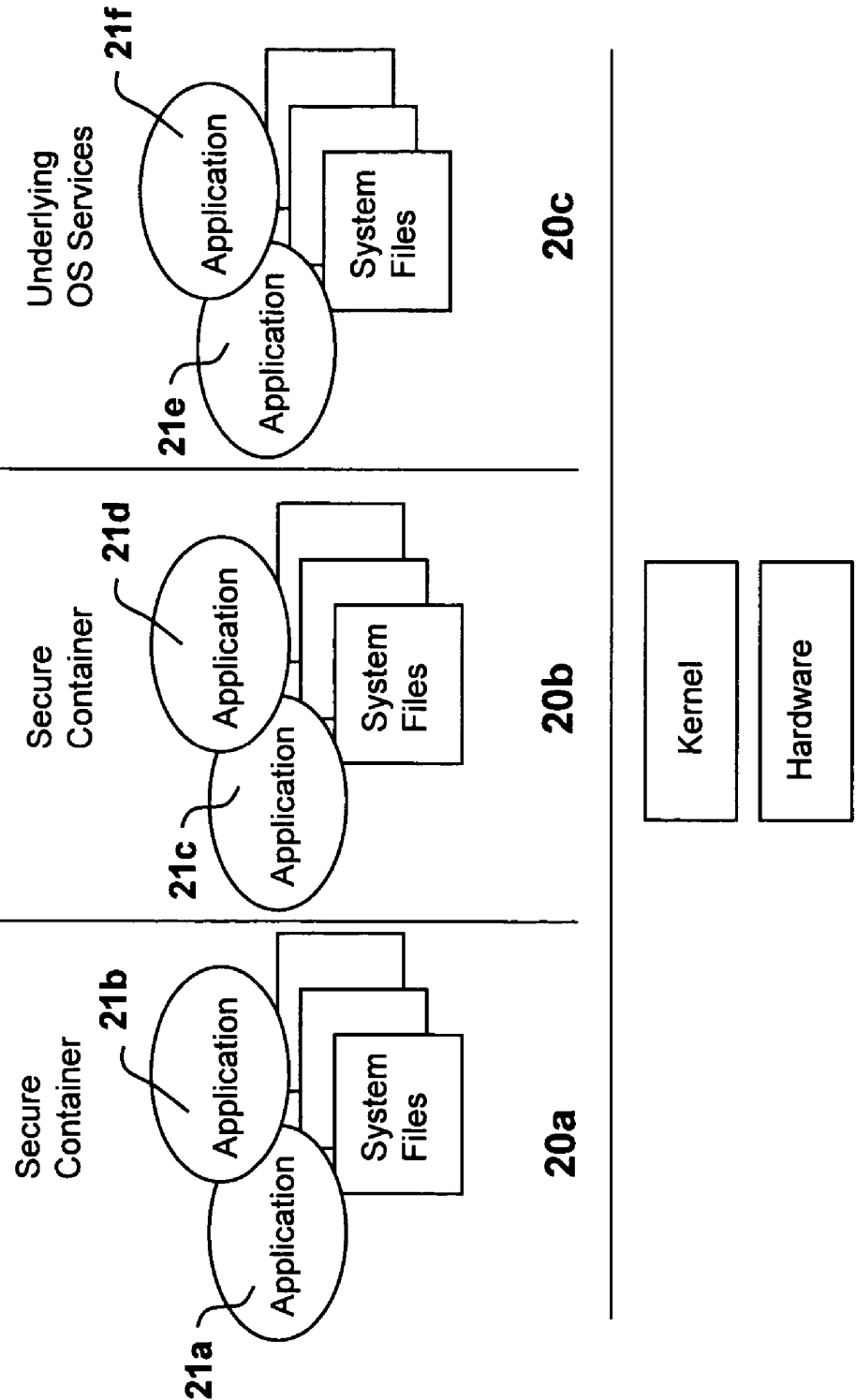


Figure 2

U.S. Patent

Apr. 14, 2009

Sheet 3 of 17

US 7,519,814 B2

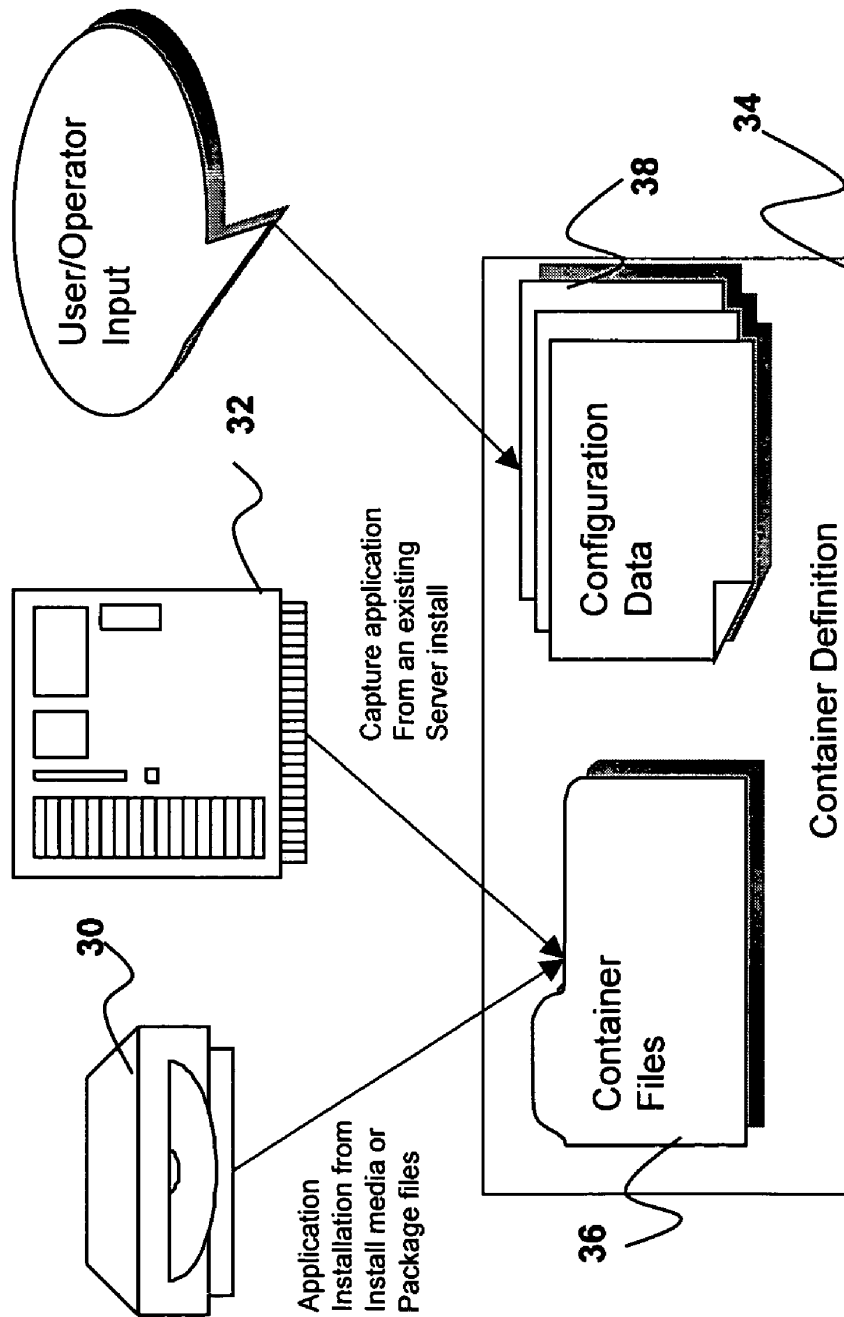


Figure 3

U.S. Patent

Apr. 14, 2009

Sheet 4 of 17

US 7,519,814 B2

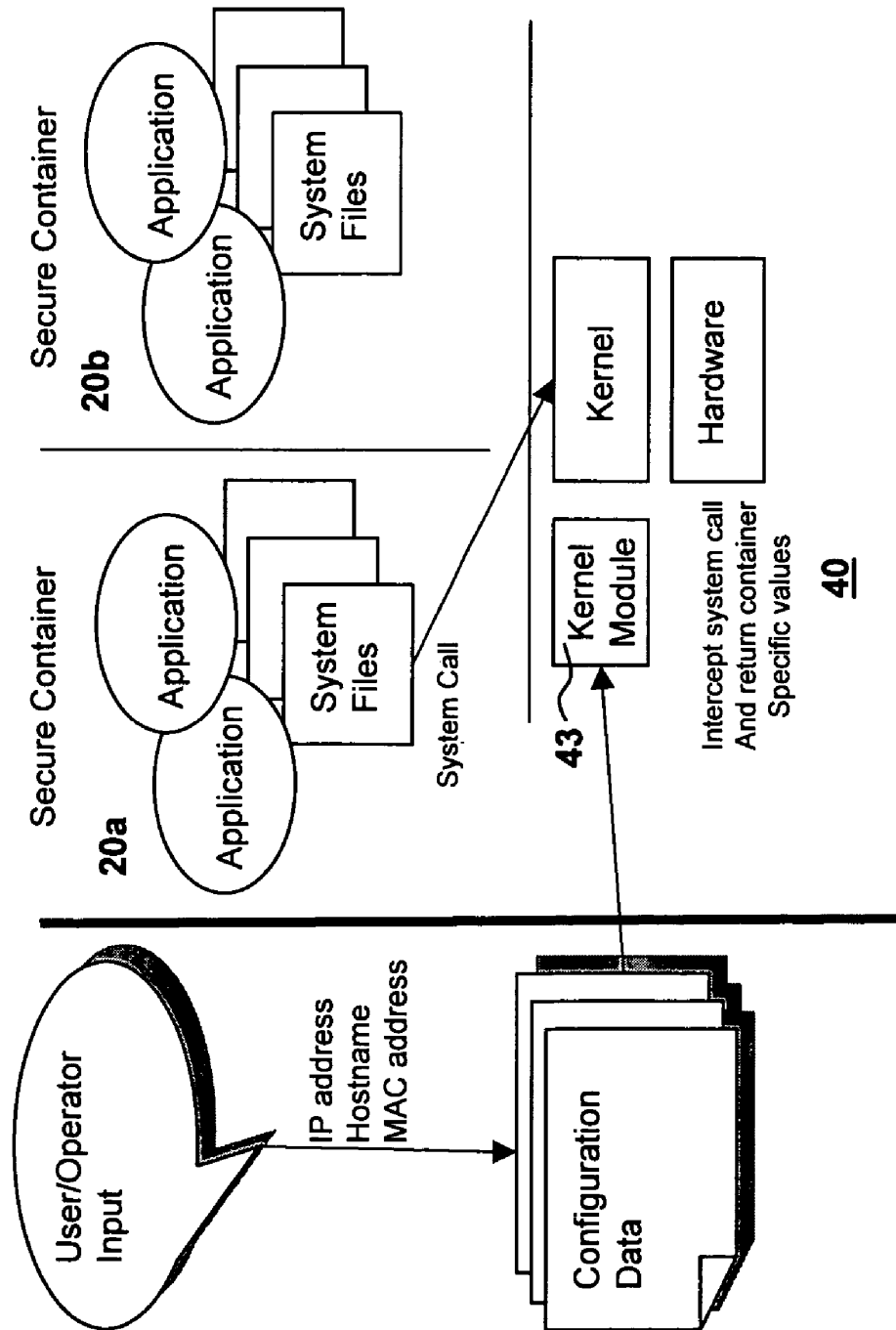


Figure 4

U.S. Patent

Apr. 14, 2009

Sheet 5 of 17

US 7,519,814 B2

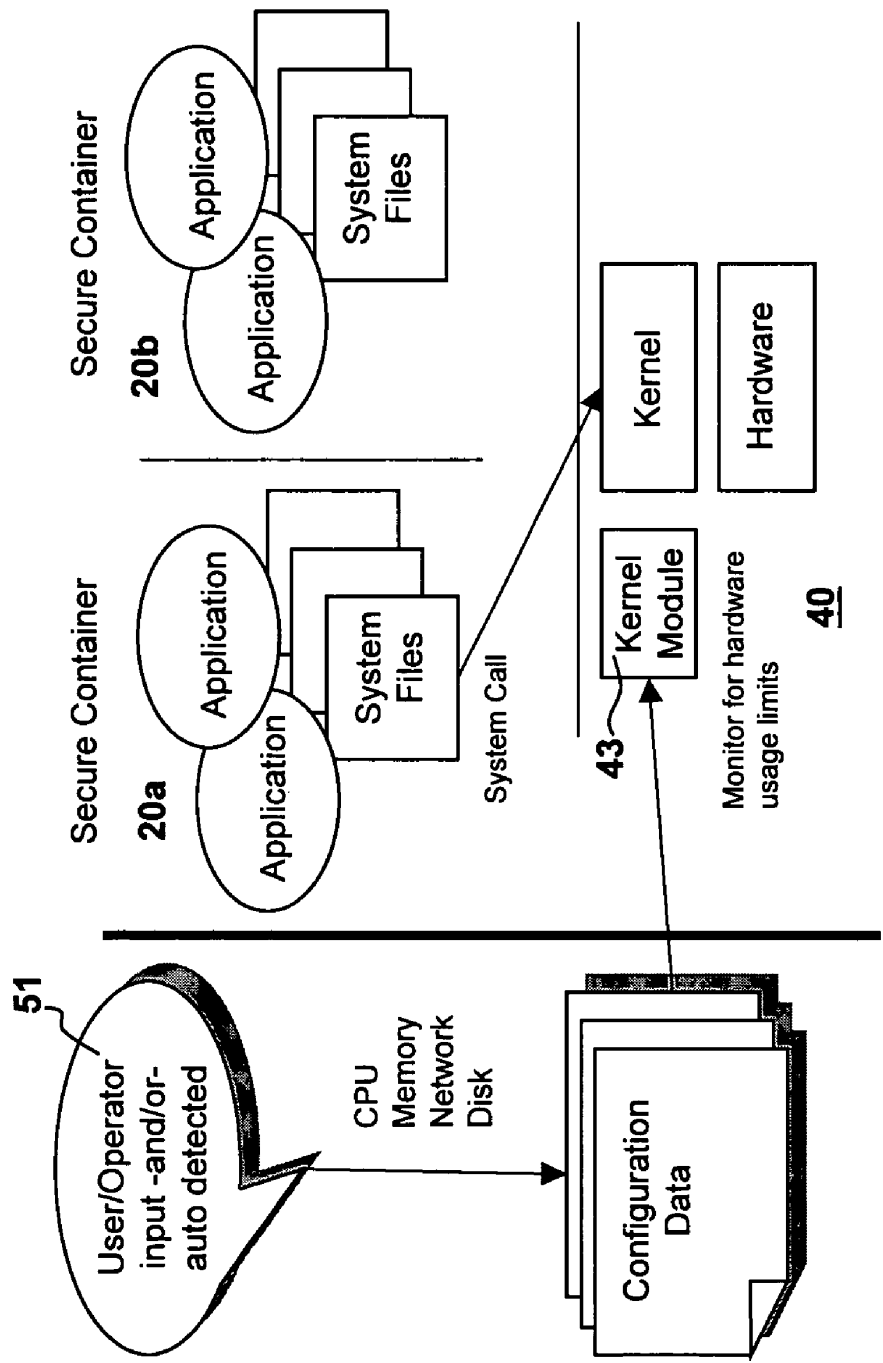


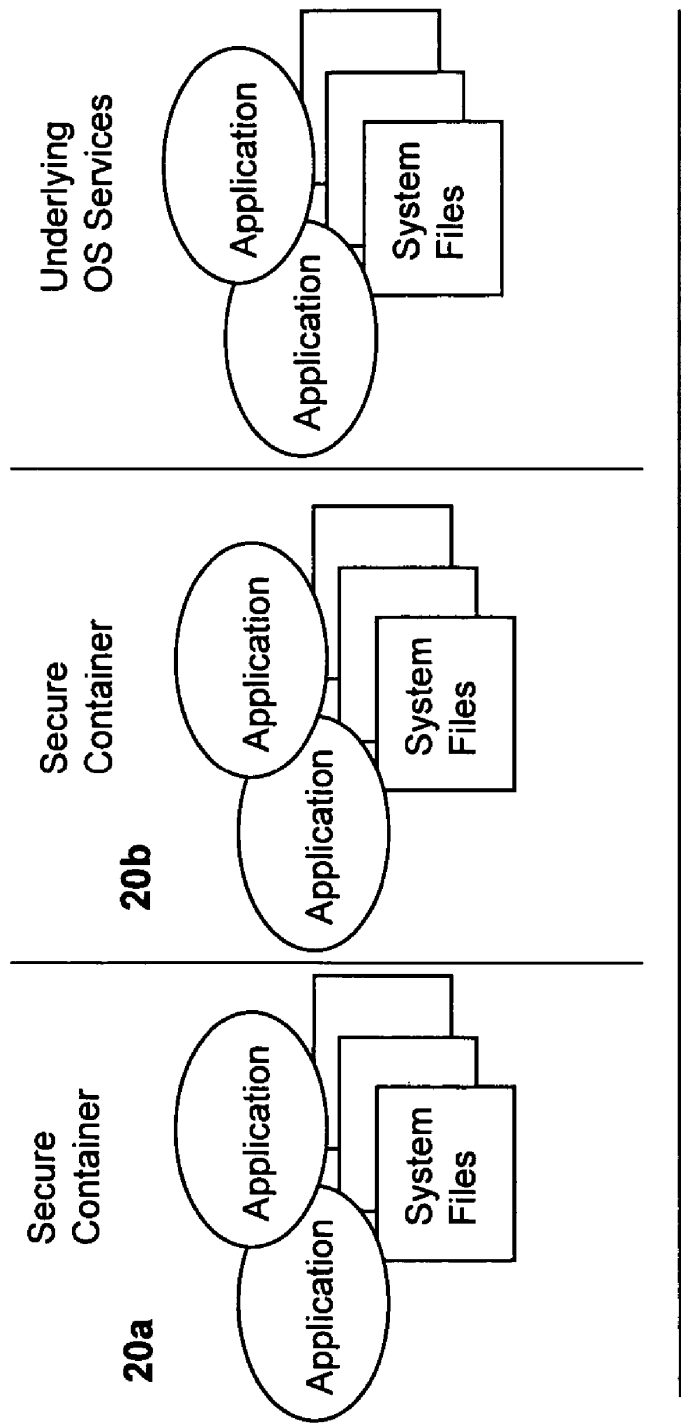
Figure 5

U.S. Patent

Apr. 14, 2009

Sheet 6 of 17

US 7,519,814 B2



40

Figure 6

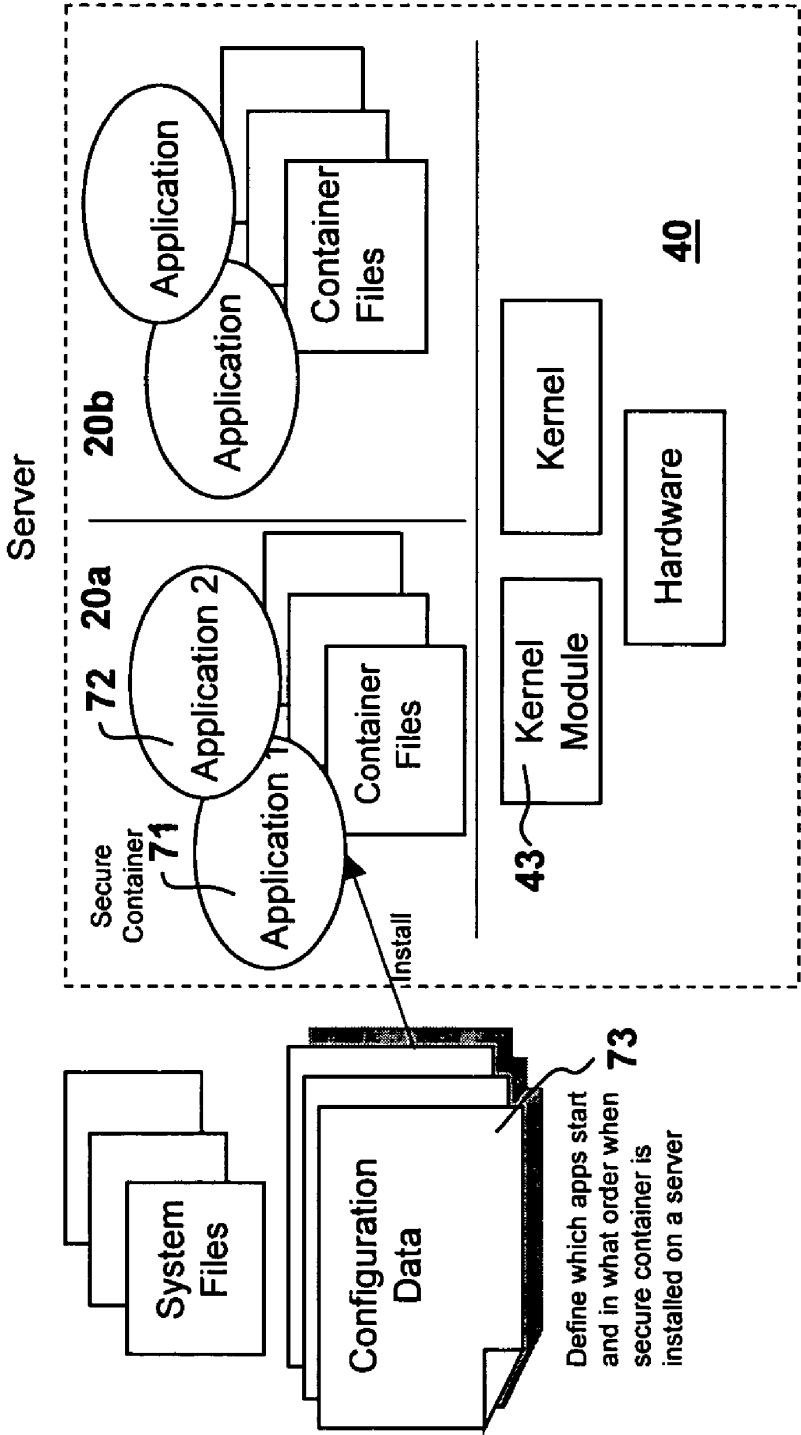


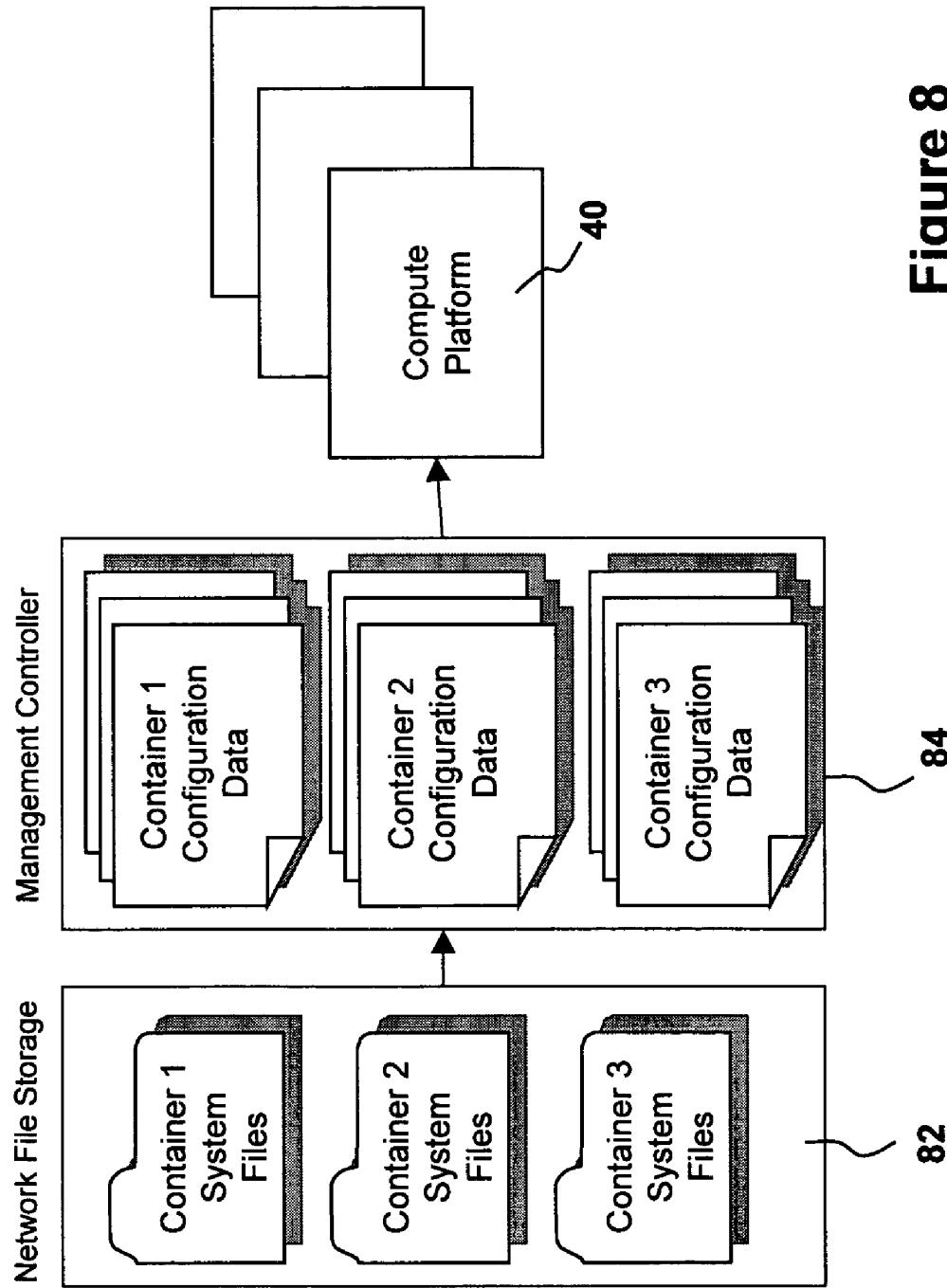
Figure 7

U.S. Patent

Apr. 14, 2009

Sheet 8 of 17

US 7,519,814 B2



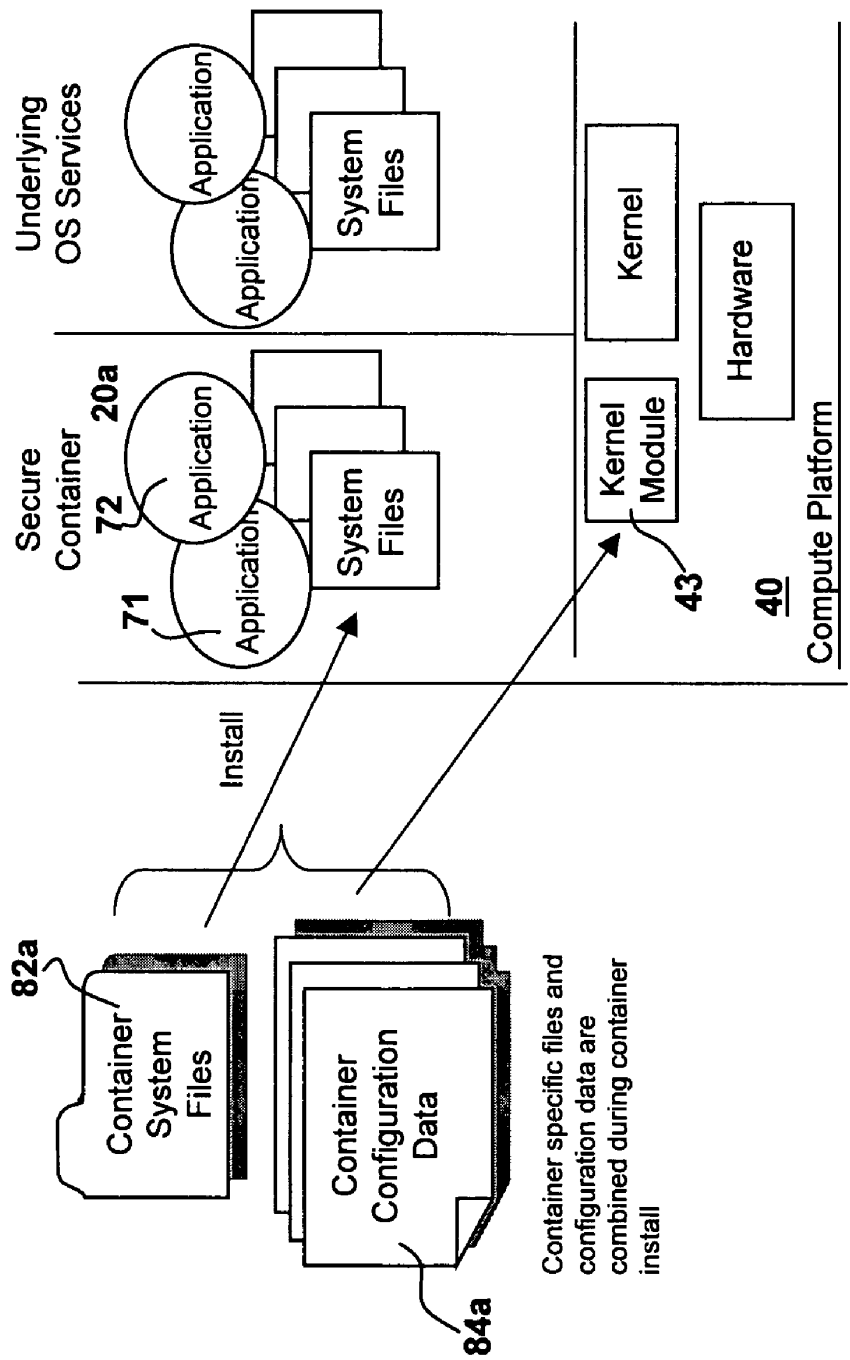


Figure 9

U.S. Patent

Apr. 14, 2009

Sheet 10 of 17

US 7,519,814 B2

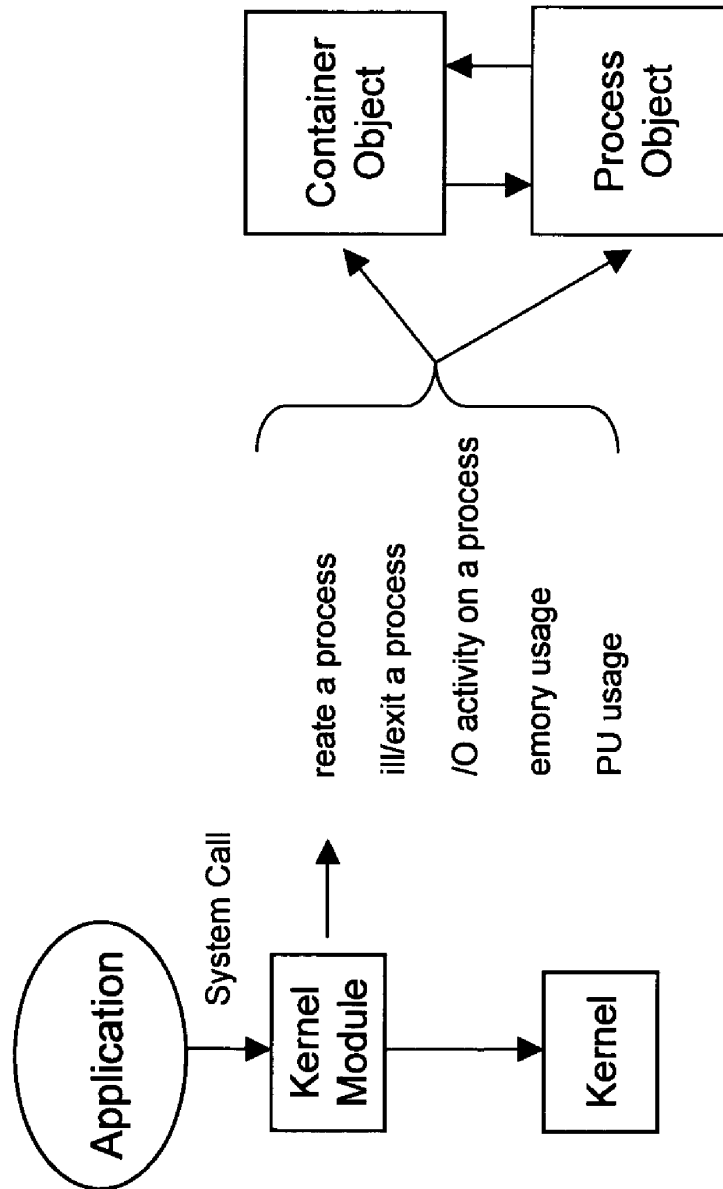


Figure 10

U.S. Patent

Apr. 14, 2009

Sheet 11 of 17

US 7,519,814 B2

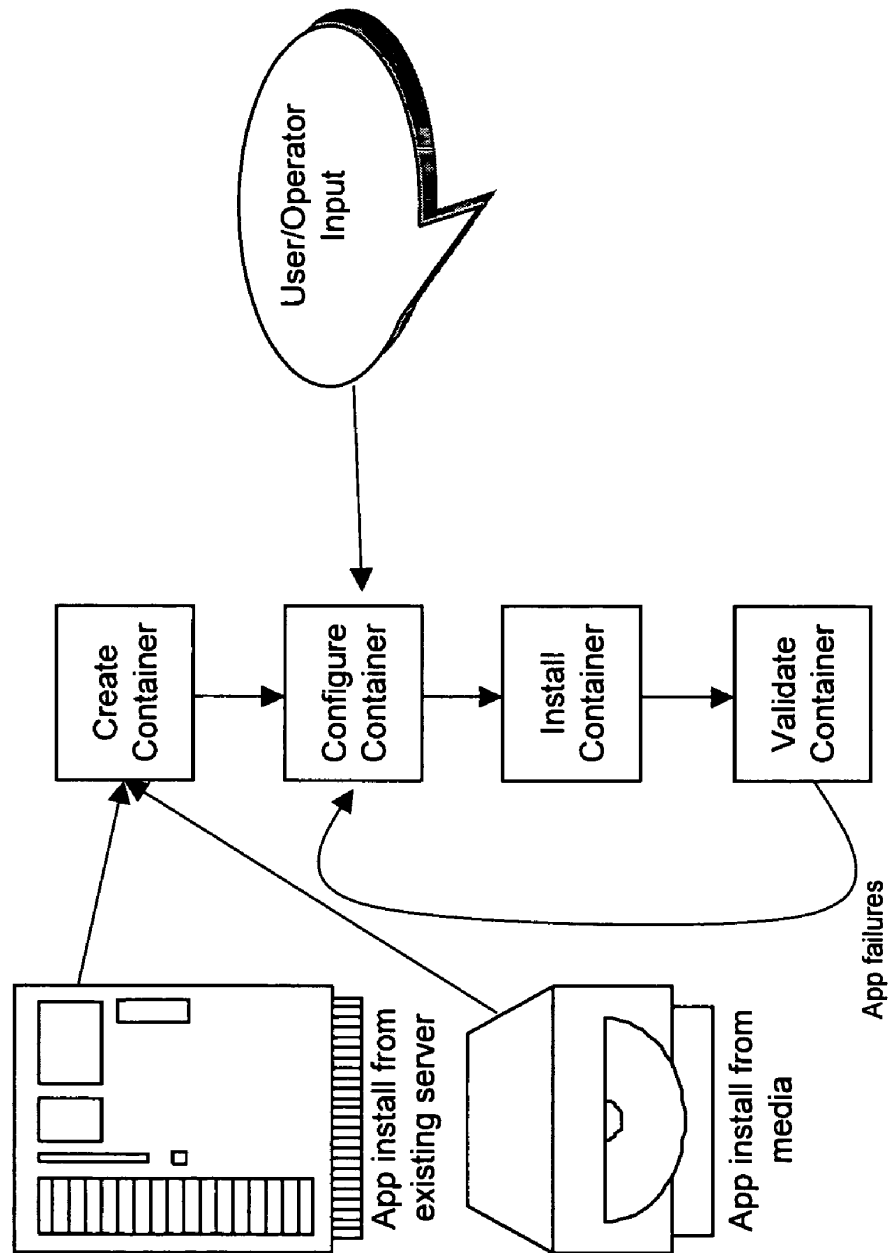


Figure 11

U.S. Patent

Apr. 14, 2009

Sheet 12 of 17

US 7,519,814 B2

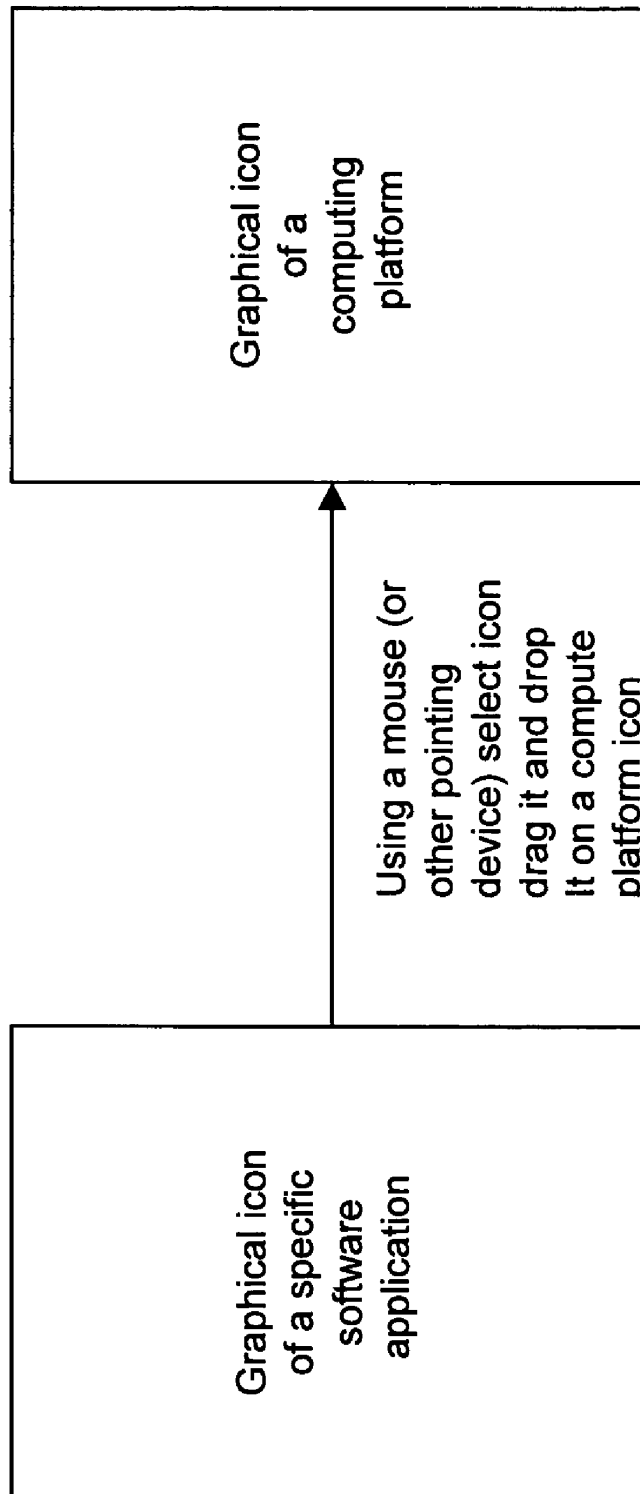


Figure 12

U.S. Patent

Apr. 14, 2009

Sheet 13 of 17

US 7,519,814 B2

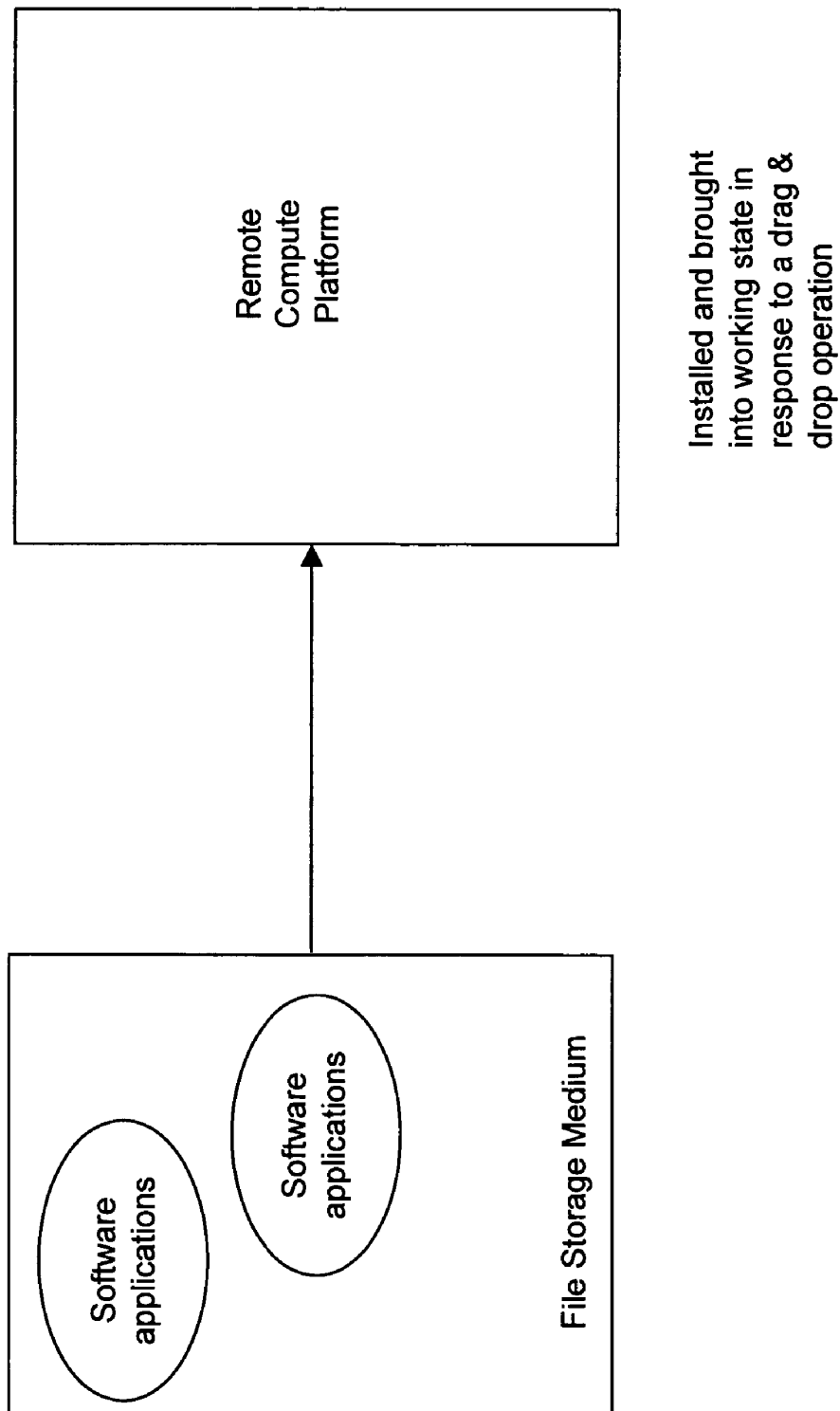


Figure 13

U.S. Patent

Apr. 14, 2009

Sheet 14 of 17

US 7,519,814 B2

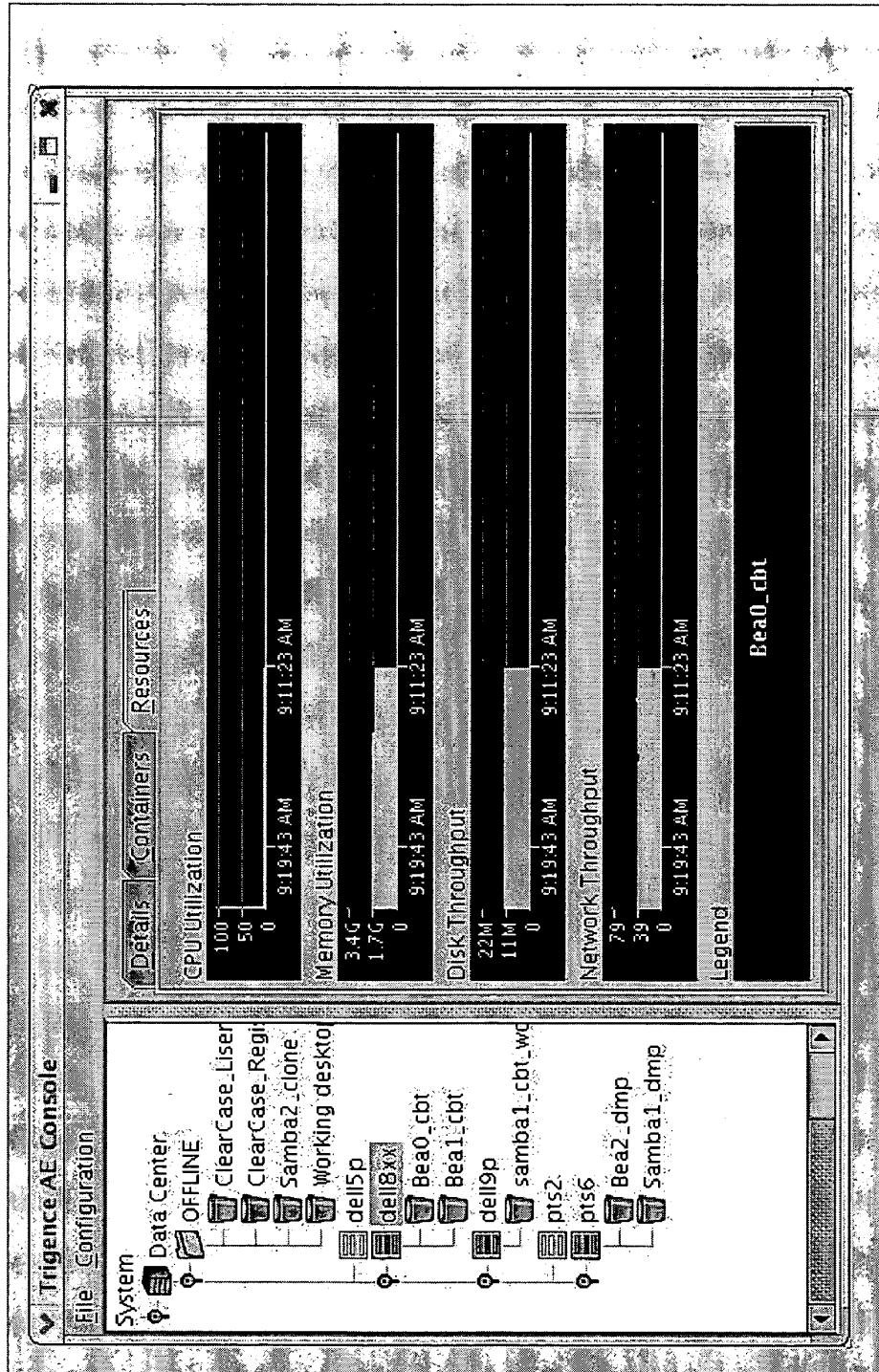


Figure 14

U.S. Patent

Apr. 14, 2009

Sheet 15 of 17

US 7,519,814 B2

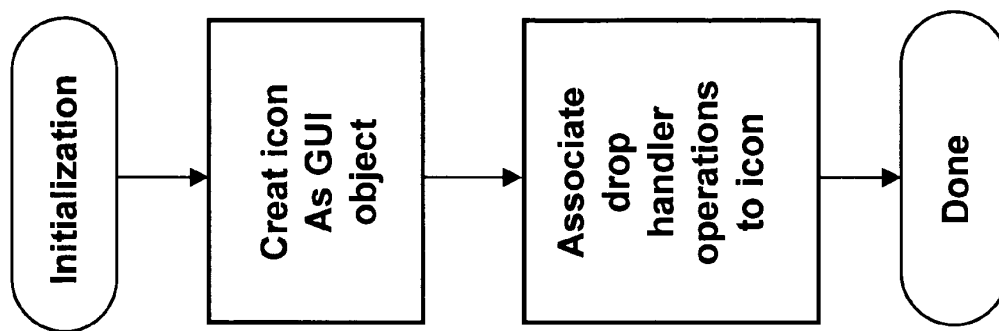


Figure 15

U.S. Patent

Apr. 14, 2009

Sheet 16 of 17

US 7,519,814 B2

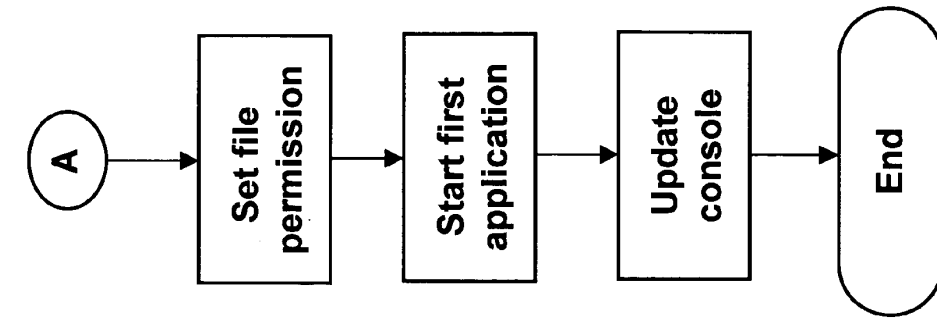
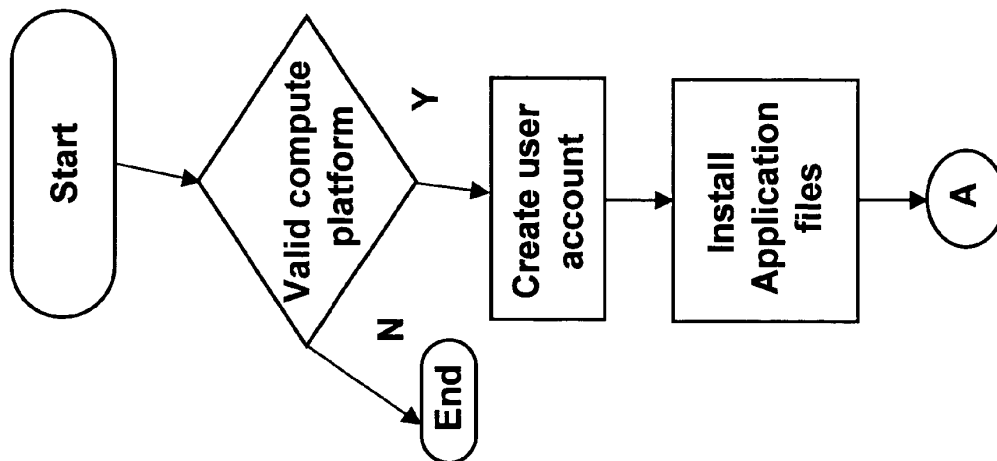


Figure 16



U.S. Patent

Apr. 14, 2009

Sheet 17 of 17

US 7,519,814 B2

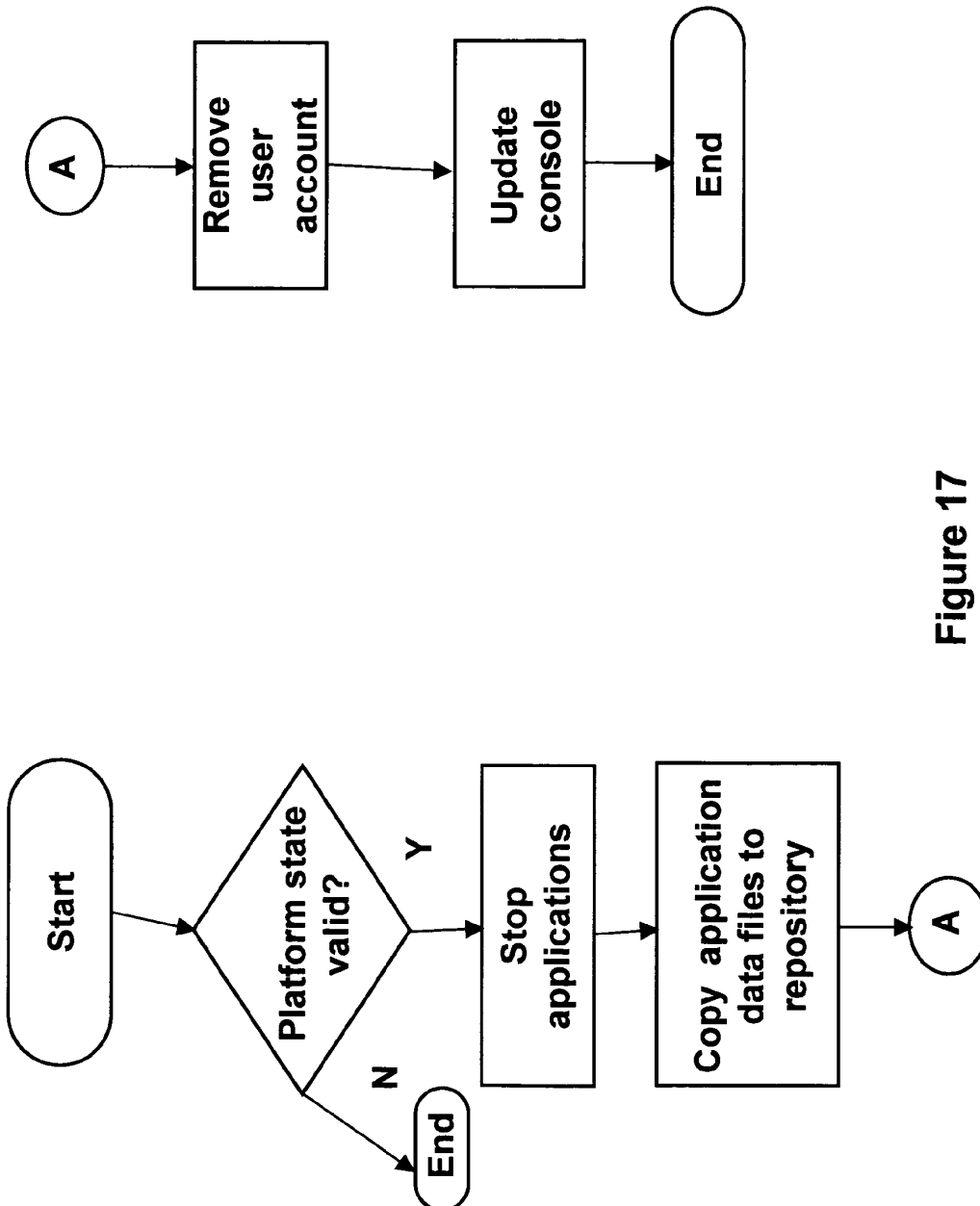


Figure 17

US 7,519,814 B2

1

**SYSTEM FOR CONTAINERIZATION OF
APPLICATION SETS****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

2

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

```
/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1
```

US 7,519,814 B2

3

/lib/libnsl.so.1
 /lib/libdl.so.2
 /lib/i686/libpthread.so.0
 /lib/i686/libc.so.6
 /lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
 /etc/httpd/conf
 /etc/httpd/conf.d
 /etc/httpd/logs
 /etc/httpd/modules
 /etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

4

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySQL and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

US 7,519,814 B2

5

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of

6

this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed “spoofing” and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. 3 is a pictorial diagram illustrating the creation of a container.

FIG. 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the “kernel module” which is used to handle system calls.

FIG. 5 is a diagram similar to FIG. 4, wherein additional configuration data is provided.

FIG. 6 is a diagram illustrating a system wherein the containers are secure containers.

FIG. 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. 9 is a diagram that illustrates the installation of a container on a server.

FIG. 10 is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. 14 is a screen snapshot of an implementation according to the schematic of FIG. 13.

FIG. 15 is a flow chart of a method of initializing icons of FIG. 14.

FIG. 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. 14; and,

FIG. 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13.

DETAILED DESCRIPTION

Turning now to FIG. 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c.

US 7,519,814 B2

7

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. 2 illustrates a system in accordance with the invention in which multiple applications **21a**, **21b**, **21c**, **21d**, **21e**, and **21f** can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20a**, **20b** and **20c**. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20a**, **20b** and **20c** are shown in FIG. 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. 4 illustrates a system having two secure containers **20a** **20b**, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20a**, **20b** environment system calls are intercepted, by the kernel module **43** passing through services installed as part of

8

this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

As introduced heretofore, “spoofing” is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system’s kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the ‘uname’ system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has ‘spoofed’ the ‘uname’ system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same “spoofing” mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

- tracking applications, for example a tracking of which applications are in and out of a specific container;
- spoofing particular values returned by the kernel;
- applying resource checks and imposing resource limits;
- monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,
- retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20a** or **20b**. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. 6 that each application executing associated with a container **20a** or **20b**, or contained within a

US 7,519,814 B2

9

container **20a** or **20b**, is able to access files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20a** association are not able to access the files contained in another container **20b**.

When a container **20a** or **20b** is installed specific applications are started, as is shown in FIG. 7 and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. 7, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. 8, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. 9 shows that the files **82a**, **84a** are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing

10

the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. 9.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform and then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. 3, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

US 7,519,814 B2

11

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

- modifying the contents a file to add a container specific IP address;
- modifying the contents of a directory to define start scripts that should be executed;
- modifying the contents of a file to define which mount points will relate to a container;
- removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications

12

are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

US 7,519,814 B2

13

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

14

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

US 7,519,814 B2

15

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. 14, a screen snapshot of an implementation is shown according to the schematic of FIG. 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. 15, shown is a flow chart of a method of initializing the icons of FIG. 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. 16 are performed by an install drop handler. During installation, verification is performed to

16

determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. 15), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

US 7,519,814 B2

17

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

18

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

- a) running an instance of a service on a server;
- b) determining which files are being used; and,
- c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim 14 comprising the steps of:

- assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
- installing the container on a server; and,
- testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

- using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with a unique icon representing

US 7,519,814 B2

19

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim 20 further comprising starting a distributed software application.

22. A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim 17 further comprising creating a user account for the service.

25. A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim 1, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;
displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

20

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim 31, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

Exhibit 2

U.S. Patent No. 7,519,814 (“814 Patent”)

Accused Instrumentalities: Amazon’s AWS End-of-Support Migration Program (“EMP”), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>To the extent the preamble is limiting, Amazon practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>AWS EMP for Windows Server enables application migration from legacy Windows Server systems to supported Windows operating systems. The EMP process includes an assessment of your application and testing requirements, packaging of the application and its dependencies on the legacy operating system based on your requirements, and migration to an AWS environment running a supported Windows Server version. When migrated to the new server environment, the application will run on the new OS as it would on the legacy operating system. The new package redirects application requests and does not include installation of any part of the legacy operating system.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p>

Claim 1	Accused Instrumentalities
	<p>EMP compatibility packages solve these challenges by resolving application dependencies on machine names, IP addresses, and ports that were created when the application was first installed on an out-of-support version of Windows Server. Network redirections are applied to packages so that the application can be migrated to a <u>new server</u>. EMP compatibility packages support Windows services common in Oracle Application Server, PeopleSoft, Tomcat, IBM WebSphere, and SQL Server 2005.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p> <p>AWS End-of-Support Migration Program (EMP) for Windows Server supports the following operating systems:</p> <ul style="list-style-type: none"> • Windows Server 2019 (64-bit) • Windows Server 2016 (64-bit) • Windows Server 2012 R2 (64-bit) • Windows Server 2008 R2 (64-bit) • Windows Server 2008 (32-bit and 64-bit) • Windows Server 2003 SP2 (32-bit) <p>The following system requirements must be met to migrate your application with AWS End-of-Support Migration Program (EMP) for Windows Server.</p> <ul style="list-style-type: none"> • .NET. Microsoft .NET 4.0 Client Profile, or later. • Memory. As required by the packaged applications. Minimum 2 GB. • Processor. As required by the packaged applications. Two CPUs recommended. • Disk space. 10 GB required to create the snapshots. The size of the EMP package depends on the size of the application. It will be 10 to 50 percent larger than the application being packaged. <p>https://docs.aws.amazon.com/emp/latest/userguide/emp-supported-os.html, Last accessed on July 13th, 2023</p> <p>Kernel-mode drivers that are a different bitness than the target operating system. Device drivers are not virtualized with EMP and therefore must be compatible with the target operating system. Compatible drivers can be deployed with the package. For example, if you are moving to a 64-bit operating system, you must have a 64-bit driver that is compatible with the new operating system.</p> <p>https://docs.aws.amazon.com/emp/latest/userguide/emp-limitations.html, Last accessed on July 13th, 2023</p>

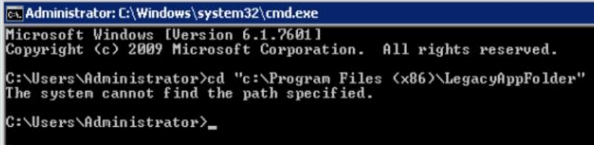
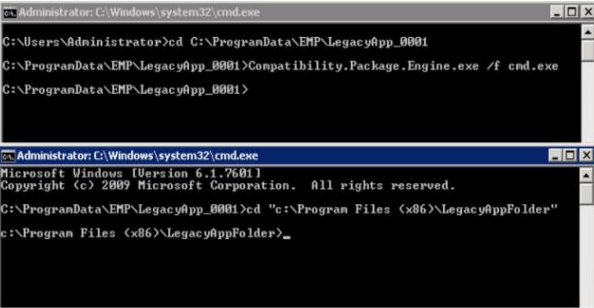
Claim 1	Accused Instrumentalities
	<p>Phase 2: Compatibility Packaging</p> <p>AWS SMEs utilize the EMP Compatibility Packager, along with the information provided in Phase 1, to identify and package applications with libraries, files, and other dependencies. The resulting package resolves OS dependencies and enables the application to successfully run on new versions of Windows Server.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p> <p>4.1.1 Application and Runtime Isolation</p> <p>Applications with conflicting requirements or outdated run times can safely run by being isolated from other applications on the server. Isolation enables EMP compatibility packages to include runtimes that conflict with other runtimes on the server, so that they are used only by the packaged application. Examples of conflicting requirements might include use of Java 1.3, .Net 2.0, and msxml.dll.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p> <p>Package an IIS-based application</p> <p>EMP supports packaging and migrating legacy IIS-based applications that run on Windows Server 2003, Windows Server 2008, and Windows Server 2008 R2 to the latest, supported versions of Windows Server running on AWS.</p> <p>https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023</p> <p>Once the package has been launched by the start of a service, when an application shortcut or other entry point into the package is launched, another log file is created within the deployed package root folder called RunWorkFlowLog.txt.</p> <p>https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023</p>

Claim 1	Accused Instrumentalities
	<p>The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p>
<p>[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>The method practiced by Amazon through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p> <p><i>See, e.g.:</i></p> <p>AWS EMP for Windows Server enables application migration from legacy Windows Server systems to supported Windows operating systems. <u>The EMP process includes an assessment of your application and testing requirements, packaging of the application and its dependencies on the legacy operating system based on your requirements, and migration to an AWS environment running a supported Windows Server version.</u> When migrated to the new server environment, <u>the application will run on the new OS as it would on the legacy operating system.</u> <u>The new package</u> redirects application requests and does not include installation of any part of the legacy operating system.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p> <p><u>The application accesses data stored in a fixed location that is not available on the new OS version:</u> the EMP engine redirects these requests to the appropriate location on the new version of the operating system.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p>

Claim 1	Accused Instrumentalities
	<p>The resulting package includes everything that the application needs to run on a modern operating system, including application files, runtimes, components, and deployment tools. The package does not include the legacy operating system, which means you never run any part of the legacy Windows Server version on the new Windows Server to which the application is upgraded.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p> <p>The configuration of a package is defined in a series of XML files, which includes:</p> <ul style="list-style-type: none"> • Configurations for Registry keys and values (AppRegistry.xml) • File, Registry, and Network redirections (Redirections.xml) • File Type Associations (FileAssociations.xml) • Shortcuts (Shortcuts.xml) • Executable programs (Programs.xml) • Environment Variables (EnvironmentVariables.xml) <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p> <p>The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p>

Claim 1	Accused Instrumentalities
	<p><u>Kernel-mode drivers</u> that are a different bitness than the target operating system. Device drivers are not virtualized with EMP and therefore must be <u>compatible with the target operating system</u>. Compatible drivers can be deployed with the package. For example, if you are moving to a 64-bit operating system, you must have a 64-bit driver that is <u>compatible with the new operating system</u>.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p>
<p>[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,</p>	<p>In the method practiced by Amazon through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.</p> <p><i>See, e.g.:</i></p> <p>AWS EMP for Windows Server enables application migration from legacy Windows Server systems to <u>supported Windows operating systems</u>. The EMP process includes an assessment of your application and testing requirements, packaging of the application and its dependencies on the legacy operating system <u>based on your requirements</u>, and migration to an AWS environment running a <u>supported Windows Server version</u>. When migrated to the new server environment, the application will run on the new OS as it would on the legacy operating system. The new package redirects application requests and does not include installation of any part of the legacy operating system.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p> <p>The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests <u>resources</u> from within the package rather than from <u>the new Windows Server OS</u>, so that the application runs successfully on <u>the new OS</u>.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p>

Claim 1	Accused Instrumentalities
	<p>Migrating legacy applications to newer, supported versions of Windows Server often requires refactoring, which is expensive and complex because of limited in-house expertise for such applications. With EMP for Windows Server, you no longer need to refactor your applications to ensure <u>compatibility with newer versions of Windows Server</u>. You now have the choice of using the tool as a self-service option at no cost or you can pay for the expert engagement to drive migration of your applications to AWS using the EMP tool.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p>
<p>[1c] the containers of application software excluding a kernel,</p>	<p>In the method practiced by Amazon through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p><i>See, e.g.:</i></p> <p>4.3.1 Application to Operating System Compatibility</p> <p>EMP software runs legacy applications within a compatibility package on modern, up-to-date operating systems. The EMP compatibility package <u>does not contain any parts of the legacy operating system</u>. Instead, it intercepts the operating system requests and redirects and <u>resolves them against the up-to-date host operating environment</u>.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023 (annotated)</p>
<p>[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by Amazon through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.</p> <p><i>See, e.g.:</i></p> <p>The package additionally includes a redirection engine that intercepts the API calls that the application makes to the underlying Windows Server OS, and redirects them to the files and registry within the created package. Therefore, the application always requests resources from within the package rather than from the new Windows Server OS, so that the application runs successfully on the new OS.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p>

Claim 1	Accused Instrumentalities
	<p>4.3.3 Services and Drivers</p> <p>EMP Compatibility packages support Windows Services out of the box. Drivers aren't captured in an EMP package but can be extracted and deployed locally using the EMP deployment script feature, as long as they are compatible with the target operating system.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p> <p>Use cmd.exe to check for the file in the local system. The system should not be able to find the file.</p>  <p>Run cmd.exe as a child process to the package engine. This should confirm that the LegacyAppFolder is available in the context of the EMP package.</p>  <p>https://docs.aws.amazon.com/emp/latest/userguide/emp-run-cmd-child.html, Last accessed on June 8th, 2023</p>
[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local	<p>In the method practiced by Amazon through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
<p>system files that remain resident on the server,</p>	<p>x64 (Engine Binaries) — Contains the EMP runtime files for packages to be deployed on a 64-bit system. When packaging is performed on a 64-bit machine, the contents of this folder are automatically copied into the root folder of the EMP package during the package build. An EMP package that was built on a 64-bit machine cannot be run on a 32-bit machine.</p> <p>x86 (Engine Binaries) — Contains the EMP runtime files for packages to be deployed on a 32-bit system. When packaging is performed on a 32-bit machine, the contents of this folder are automatically copied into the root folder of the EMP package during the package build. An EMP package that was built on a 32-bit machine will run on a 64-bit machine, however, we recommend that you use the appropriate runtimes for the destination platform. This is automatically handled during the deployment process.</p> <p>https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023</p> <p>Applications with conflicting requirements or outdated run times can safely run by being isolated from other applications on the server. Isolation enables EMP compatibility packages to include runtimes that conflict with other runtimes on the server, so that they are used only by the packaged application. Examples of conflicting requirements might include use of Java 1.3, .Net 2.0, and msxml.dll.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p>
<p>[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,</p>	<p>In the method practiced by Amazon through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>EMP software runs legacy applications within a compatibility package on modern, up-to-date operating systems. The EMP compatibility package does not contain any parts of the legacy operating system. Instead, it intercepts the operating system requests and redirects and resolves them against the up-to-date host operating environment.</p> <p>The EMP compatibility package permits the legacy application to run alongside other applications and/or other versions of the same application. For example, multiple versions of Microsoft Office can run simultaneously on the target operating system, or two incompatible 32-bit applications can run together, isolated from each other by the EMP compatibility package.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p> <p>Applications with conflicting requirements or outdated run times can safely run by being isolated from other applications on the server. Isolation enables EMP compatibility packages to include runtimes that conflict with other runtimes on the server, so that they are used only by the packaged application. Examples of conflicting requirements might include use of Java 1.3, .Net 2.0, and msxml.dll.</p> <p>https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf Last accessed on June 2nd, 2023</p>
[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.	<p>In the method practiced by Amazon through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>This section describes the folders and files that are included in an EMP compatibility package. When the EMP compatibility packaging process is complete, the output of the package builder is called an EMP compatibility package. The package contains both the file and registry data of the packaged application, and the EMP binaries and configuration files that are required to deploy and run the packaged application.</p> <p>The EMP package, which is the product to deploy, is called the source package. The post-deployment package is called the deployed package. These packages are slightly different at the level of the root package folder. However, the packaged application content is the same in both packages.</p> <p>https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023</p> <p>x64 (Engine Binaries) — Contains the EMP runtime files for packages to be deployed on a 64-bit system. When packaging is performed on a 64-bit machine, the contents of this folder are automatically copied into the root folder of the EMP package during the package build. An EMP package that was built on a 64-bit machine cannot be run on a 32-bit machine.</p> <p>x86 (Engine Binaries) — Contains the EMP runtime files for packages to be deployed on a 32-bit system. When packaging is performed on a 32-bit machine, the contents of this folder are automatically copied into the root folder of the EMP package during the package build. An EMP package that was built on a 32-bit machine will run on a 64-bit machine, however, we recommend that you use the appropriate runtimes for the destination platform. This is automatically handled during the deployment process.</p> <p>https://docs.aws.amazon.com/pdfs/emp/latest/userguide/service-guide.pdf Last accessed on June 2nd, 2023</p>

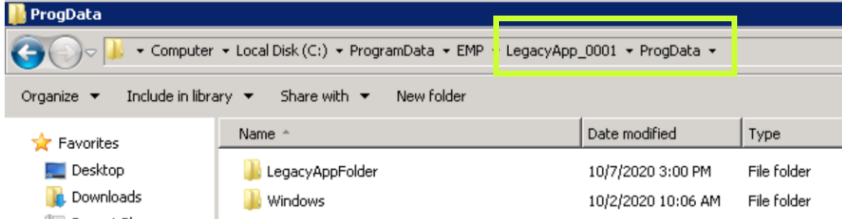
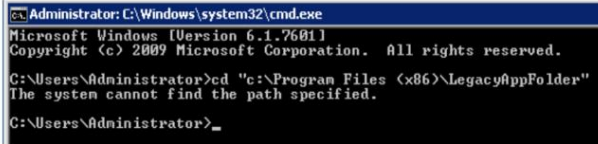
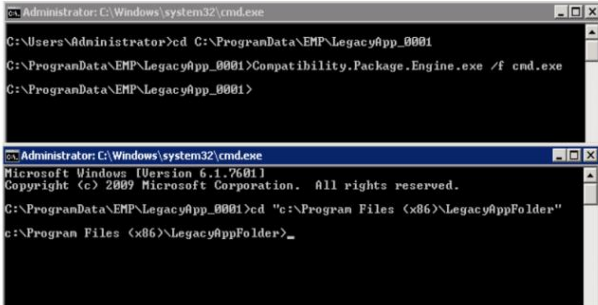
Claim 1	Accused Instrumentalities
	 <p>https://docs.aws.amazon.com/emp/latest/userguide/emp-run-cmd-child.html, Last accessed on June 8th, 2023 (annotated)</p> <p>Use cmd.exe to check for the file in the local system. The system should not be able to find the file.</p>  <p>Run cmd.exe as a child process to the package engine. This should confirm that the LegacyAppFolder is available in the context of the EMP package.</p>  <p>https://docs.aws.amazon.com/emp/latest/userguide/emp-run-cmd-child.html, Last accessed on June 8th, 2023</p>

Exhibit 3



(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,784,058 B2**
(45) **Date of Patent:** **Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES**

2002/0004854 A1 1/2002 Hartley
2002/0174215 A1 11/2002 Schaefer 709/224
2003/0101292 A1 5/2003 Fisher
2004/0025165 A1* 2/2004 Desoli et al. 719/310

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul
O’Leary**, Kanata (CA); **Dean Huffman**,
Kanata (CA)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002
WO WO 2006/039181 A 4/2006

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

OTHER PUBLICATIONS

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 1293 days.

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

* cited by examiner

(21) Appl. No.: **10/946,536**

Primary Examiner—Hyung S Sough

Assistant Examiner—Syed Roni

(22) Filed: **Sep. 21, 2004**

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt,
Milbrath & Gilchrist, P.A. Attorneys at Law

(65) **Prior Publication Data**

US 2005/0066303 A1 Mar. 24, 2005

(57) **ABSTRACT**

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep.
22, 2003.

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

(51) **Int. Cl.**
G06F 9/22 (2006.01)

(52) **U.S. Cl.** **719/310; 719/319**

(58) **Field of Classification Search** 719/310,
719/319

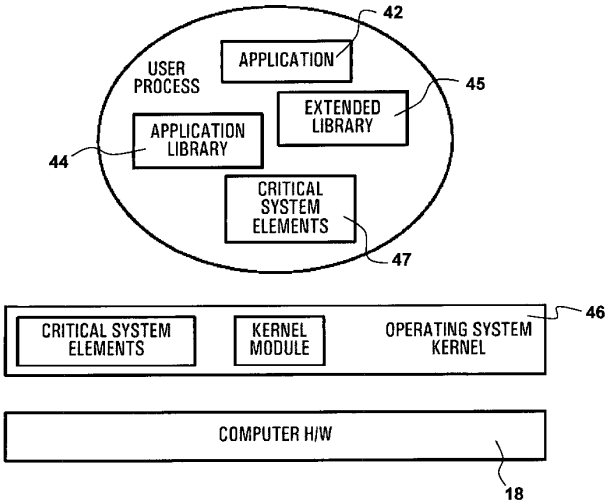
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,481,706 A * 1/1996 Peek 710/200
6,212,574 B1 * 4/2001 O’Rourke et al. 719/321
6,260,075 B1 * 7/2001 Cabrero et al. 719/310

18 Claims, 7 Drawing Sheets



U.S. Patent

Aug. 24, 2010

Sheet 1 of 7

US 7,784,058 B2

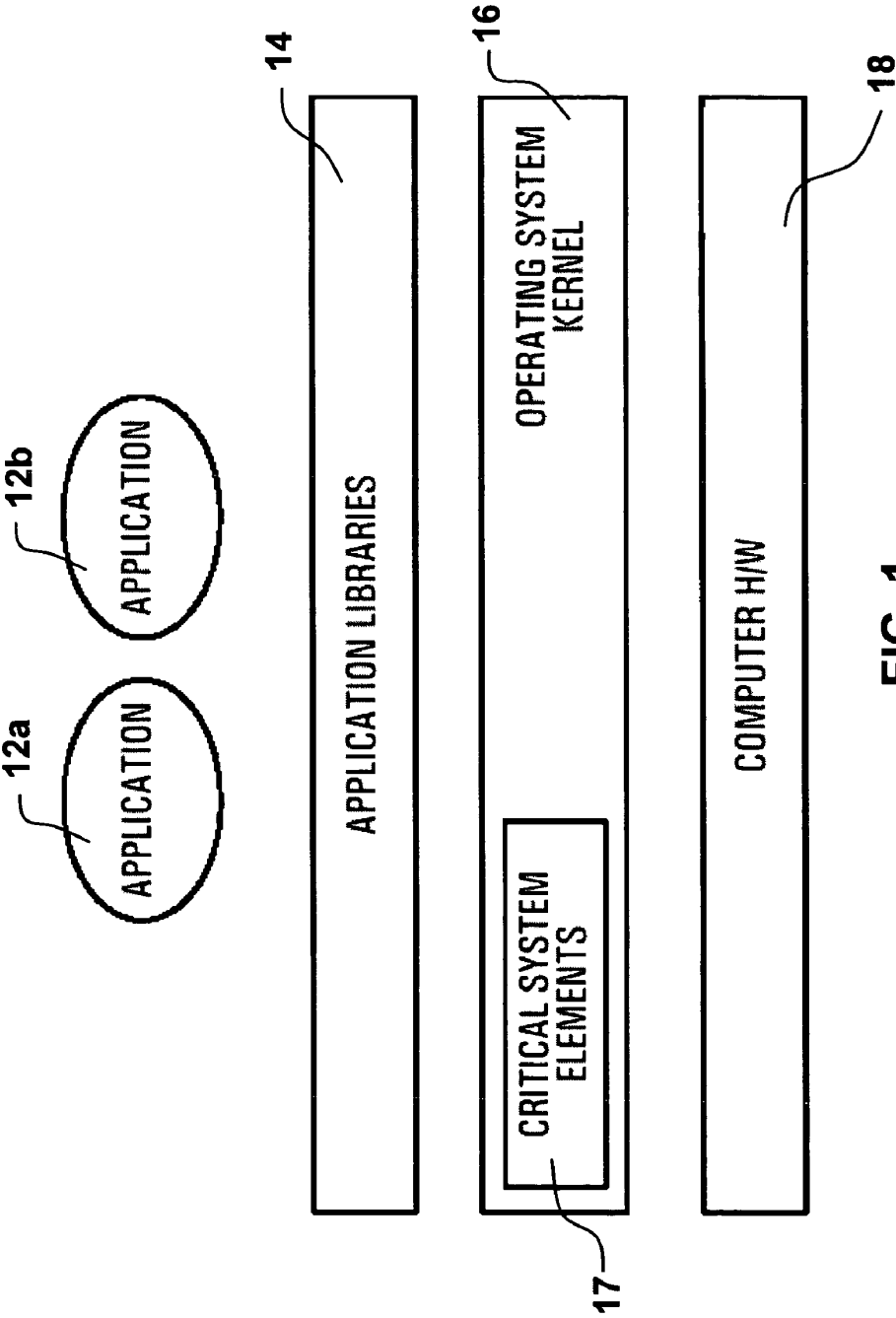


FIG. 1
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 2 of 7

US 7,784,058 B2

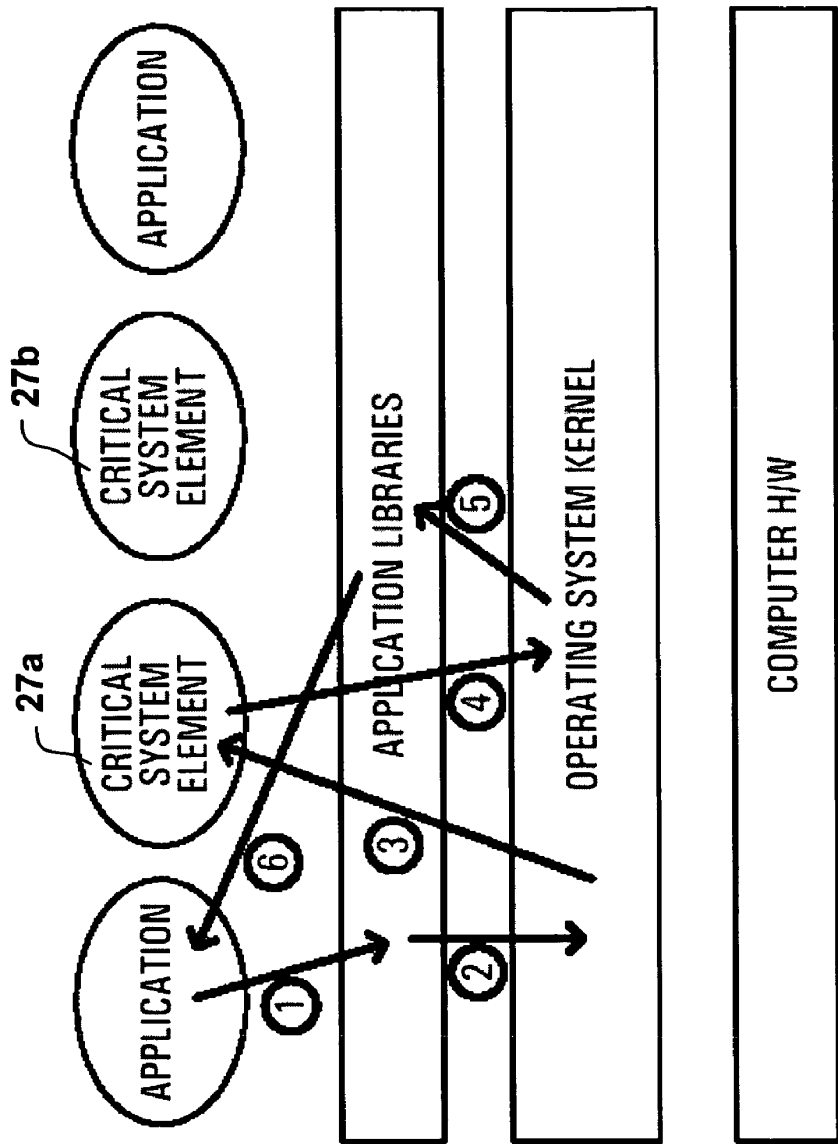


FIG. 2a
Prior Art

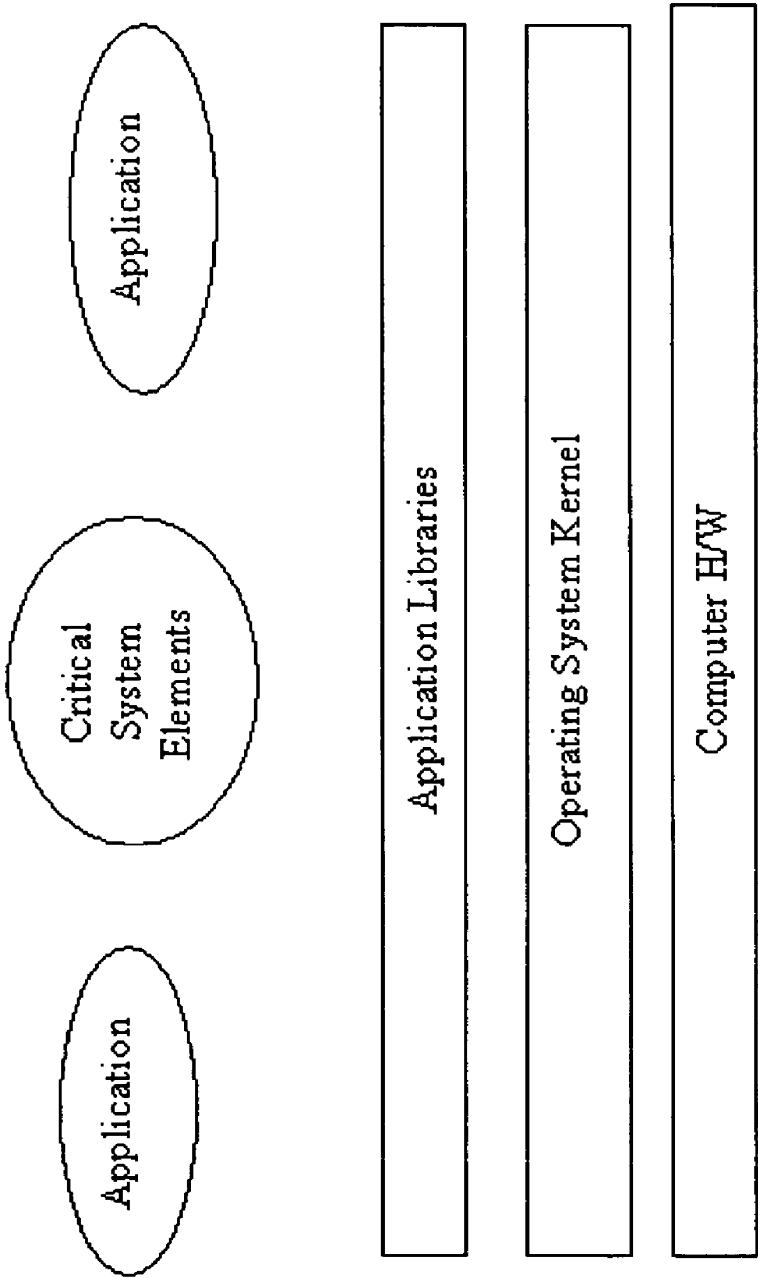


FIG. 2b
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 4 of 7

US 7,784,058 B2

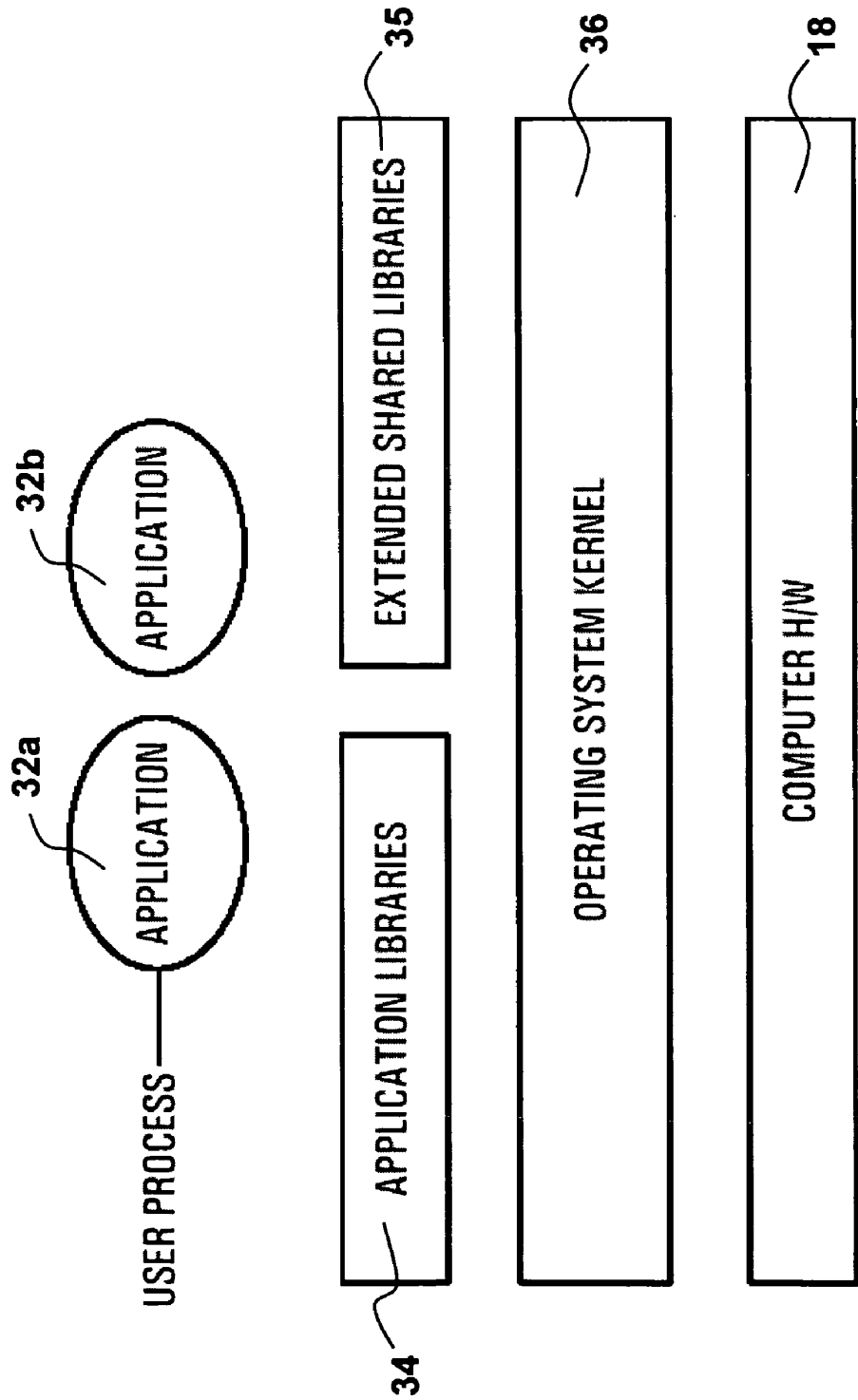


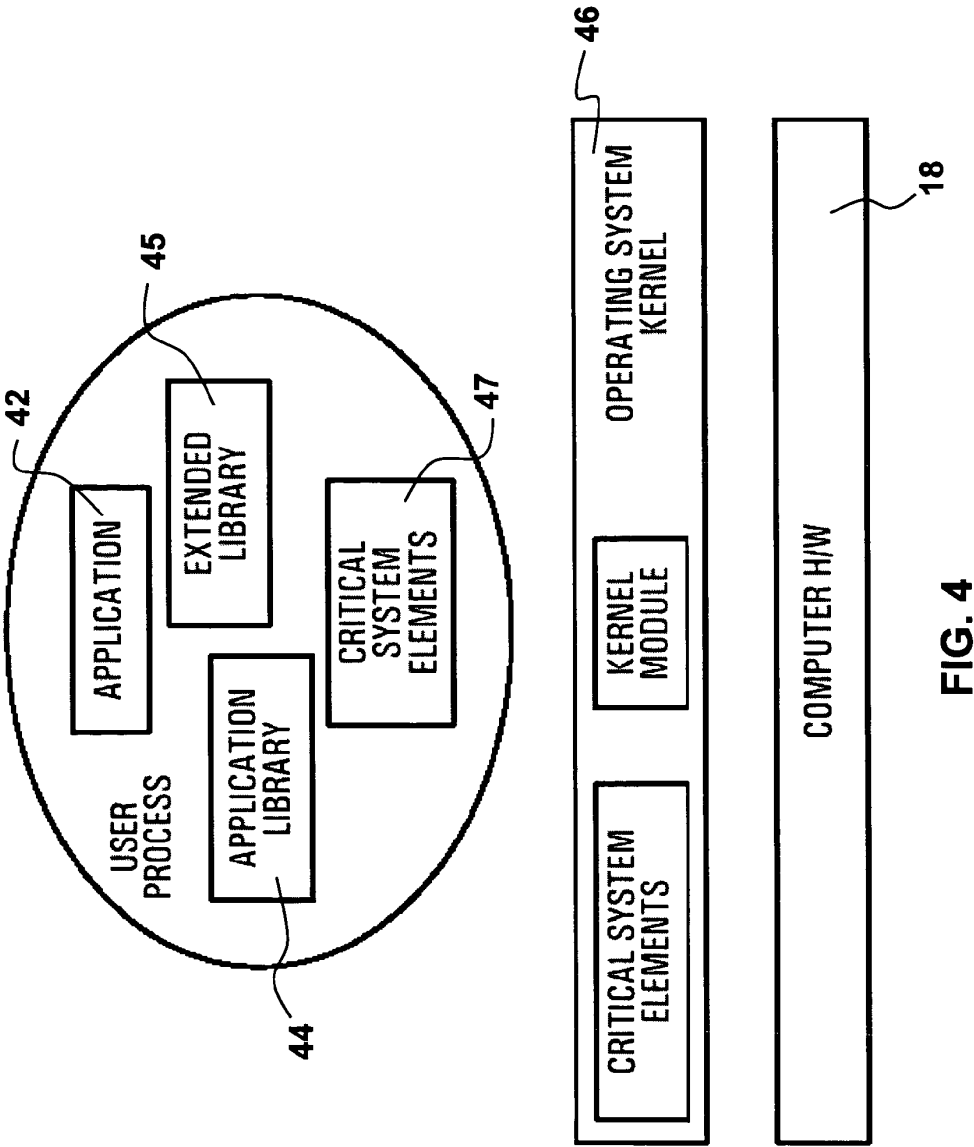
FIG. 3

U.S. Patent

Aug. 24, 2010

Sheet 5 of 7

US 7,784,058 B2



U.S. Patent

Aug. 24, 2010

Sheet 6 of 7

US 7,784,058 B2

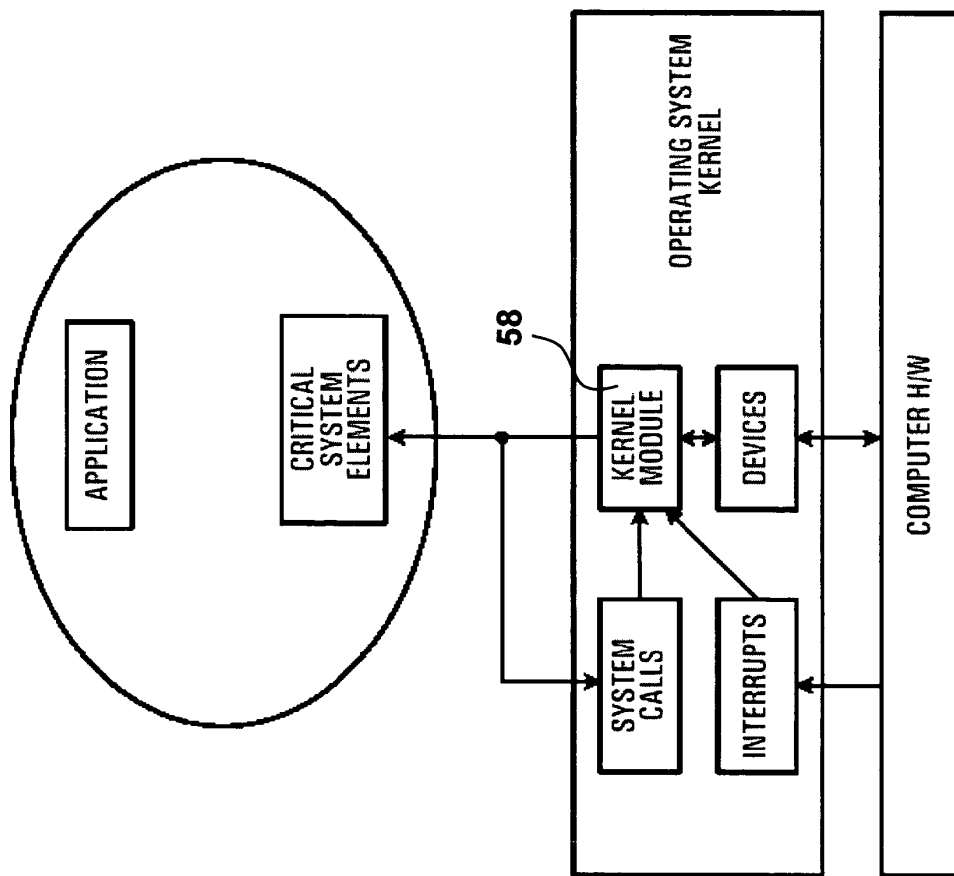


FIG. 5

U.S. Patent

Aug. 24, 2010

Sheet 7 of 7

US 7,784,058 B2

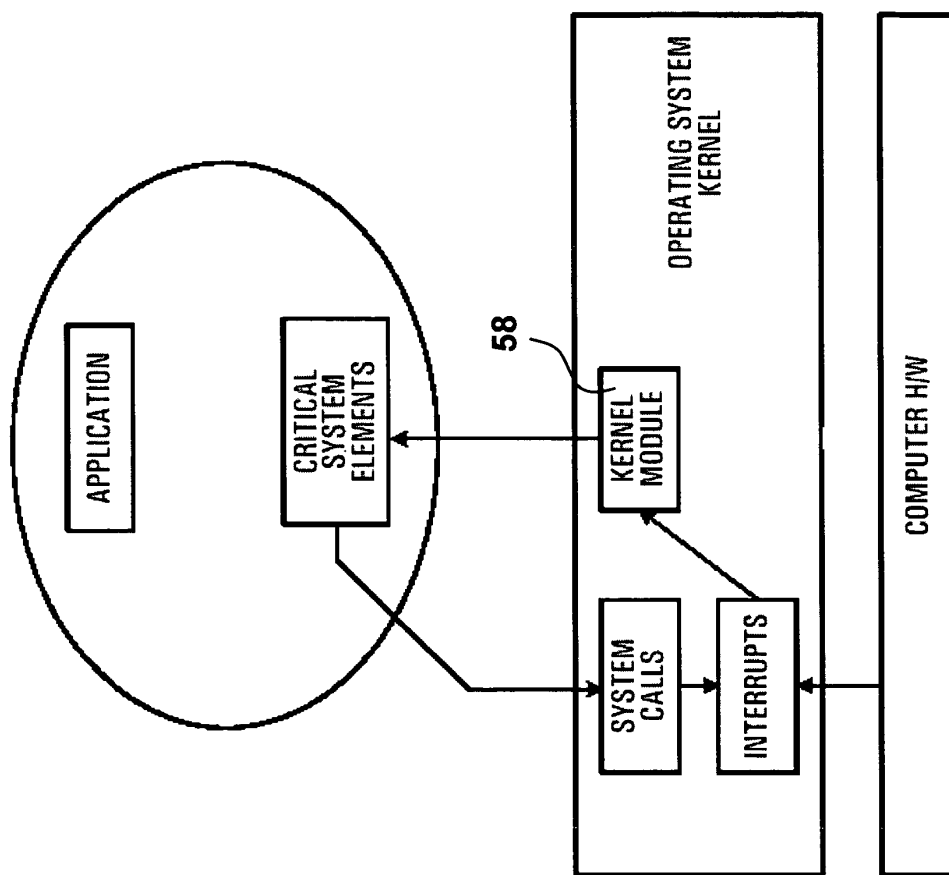


FIG. 6

US 7,784,058 B2

1

**COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled “User Mode Critical System Element as Shared Libs”, which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

2

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.

2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

US 7,784,058 B2

3

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

4

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. 1 is an architectural view of the traditional monolithic prior art operating system;

US 7,784,058 B2

5

FIG. 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. 3 is an architectural view of an embodiment of the invention;

FIG. 4 is a functional view showing how critical system elements exist in the same context as an application;

FIG. 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. 6 shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TCP/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP

6

stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality

2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided

2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b

US 7,784,058 B2

7

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12a**, **12b** makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12a**, **12b** in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2a** shows a system architecture where critical system elements **27a**, **27b** execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2a** is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2a** as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2b** is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2a** and FIG. **2b** is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2b** is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the

8

case with a multi-server architecture depicted in FIG. **2a**. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32a**, **32b**, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45,47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

US 7,784,058 B2

9

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. 4. FIG. 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

10

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:
 - a) a processor;
 - b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,
 - c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

US 7,784,058 B2

11

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

6. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

7. A computing system according to claim 6 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

8. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

12

9. A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

10. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

11. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

12. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

13. A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

14. A computing system according to claim 13 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

15. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

16. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

17. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.

* * * * *

Exhibit 4

U.S. Patent No. 7,519,814 (“’814 Patent”)

Accused Instrumentalities: the Amazon Elastic Container Service (“ECS”), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

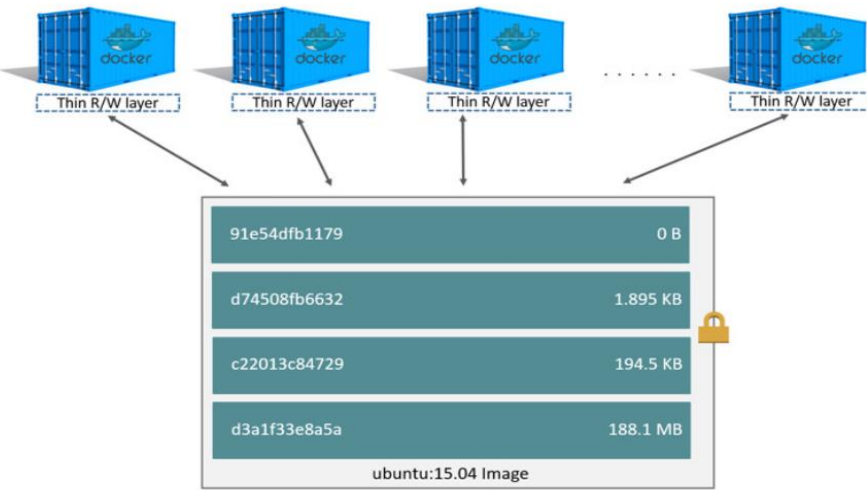
Claim 1	Accused Instrumentalities
[1pre] 1. A computing system for executing a plurality of software applications comprising:	<p>To the extent the preamble is limiting, each Accused Instrumentality comprise or constitute a computing system for executing a plurality of software applications as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>Amazon ECS capacity</p> <p>Amazon ECS capacity is the infrastructure where your containers run. https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023</p> <p>There are three layers in Amazon ECS:</p> <ul style="list-style-type: none">• Capacity - The infrastructure where your containers run• Controller - Deploy and manage your applications that run on the containers• Provisioning - The tools that you can use to interface with the scheduler to deploy and manage your applications and containers <p>https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023</p>

Claim 1	Accused Instrumentalities
	<p>To deploy applications on Amazon ECS, your application components must be configured to run in <i>containers</i>. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an <i>image</i>. Images are typically built from a Dockerfile. A Dockerfile is a plaintext file that specifies all of the components that are included in the container. After they're built, these images are stored in a <i>registry</i> such as Amazon ECR where they can be downloaded from.</p> <p>https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023</p>
[1a] a) a processor;	<p>Each Accused Instrumentality comprises a processor.</p> <p><i>See, e.g.:</i></p> <p>Amazon ECS supports using 64-bit ARM applications. You can run your applications on the platform that's powered by AWS Graviton2 processors,. It's suitable for a wide variety of workloads. This includes workloads such as application servers, micro-services, high-performance computing, CPU-based machine learning inference, video encoding, electronic design automation, gaming, open-source databases, and in-memory caches.</p> <p>https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 (annotated)</p>
[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,	<p>Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.</p> <p><i>See, e.g.:</i></p>

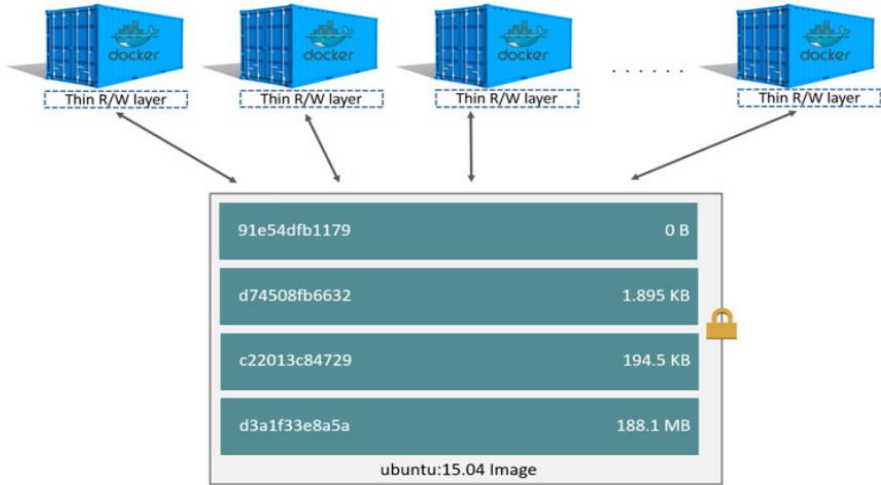
Claim 1	Accused Instrumentalities
	<p>An Amazon ECS container instance is an Amazon EC2 instance that is running the Amazon ECS container agent and has been registered into an Amazon ECS cluster. When you run tasks with Amazon ECS using the EC2 launch type or an Auto Scaling group capacity provider, your tasks are placed on your active container instances.</p> <p>Note Tasks using the Fargate launch type are deployed onto infrastructure managed by AWS, so this topic does not apply.</p> <p>The following Linux container instance <u>operating systems</u> are available:</p> <ul style="list-style-type: none"> • <u>Amazon Linux</u>: This is a general purpose operating system. • Bottlerocket: This is an operating system that is optimized for container workloads and that has a focus on security. It does not include a package manager and is immutable by default. For information about the security features and guidance, see Security Features and Security Guidance on the GitHub website. <p>An Amazon ECS container instance specification consists of the following components:</p> <p>Required</p> <ul style="list-style-type: none"> • A modern Linux distribution running at least version 3.10 of the Linux kernel. • The Amazon ECS container agent (preferably the latest version). For more information, see Amazon ECS container agent (p. 357). <p>https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023 (annotated)</p>
[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and	<p>Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.</p> <p><i>See, e.g.:</i></p>

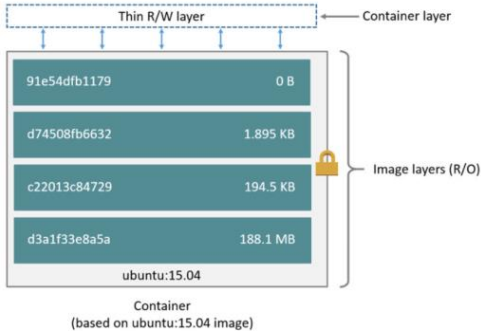
Claim 1	Accused Instrumentalities
	<p>To deploy applications on Amazon ECS, your application components must be configured to run in <i>containers</i>. A container is a standardized unit of software development that holds everything that your software application requires to run. This includes relevant code, runtime, system tools, and system libraries. Containers are created from a read-only template that's called an <i>image</i>. Images are typically built from a Dockerfile. A Dockerfile is a plaintext file that specifies all of the components that are included in the container. After they're built, these images are stored in a <i>registry</i> such as Amazon ECR where they can be downloaded from.</p> <p>https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023</p> <p>Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see Deploying Docker containers on Amazon ECS.</p> <p>https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023</p> <p>A Docker image is a read-only template that defines your container. The image contains the code that will run including any definitions for any libraries and dependencies your code needs. A Docker container is an instantiated (running) Docker image. AWS provides Amazon Elastic Container Registry (ECR), an image registry for storing and quickly retrieving Docker images.</p> <p>https://aws.amazon.com/docker/#, Last accessed on June 14, 2023</p>
[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of	<p>In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.</p> <p>For example, a Docker base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and</p>

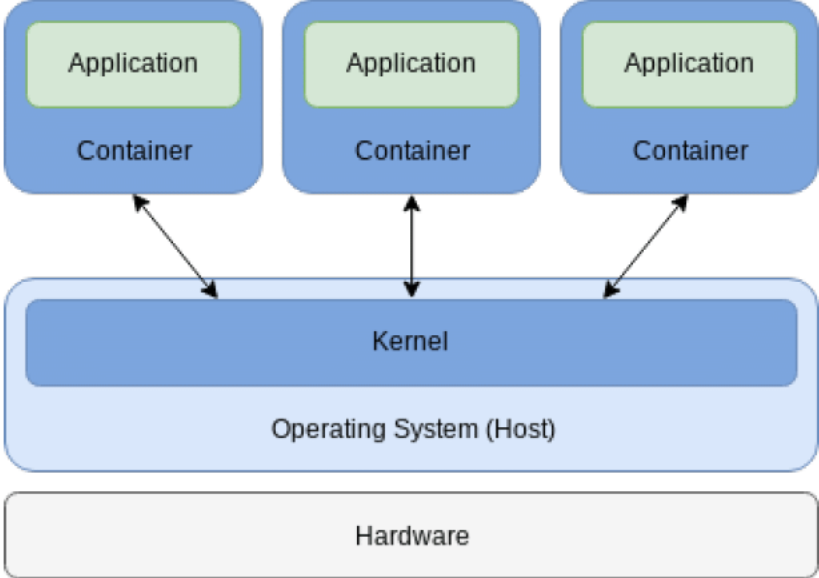
Claim 1	Accused Instrumentalities
the one or more of the plurality of software applications,	<p>dependencies, including system libraries or shared library critical system elements (SLCSEs). When the container runs the image, it creates a runtime instance of that container image.</p> <p><i>See, e.g.:</i></p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image, Last accessed on June 14, 2023</p>

Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/, Last accessed on June 14, 2023</p>
<p>[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software</p>	<p>In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.</p> <p>When a Docker image is used to create a container, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications</p>

Claim 1	Accused Instrumentalities
<p>applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and</p>	<p>running in their respective containers. The docker image includes essential system files, libraries, and dependencies required to run the software application within the container. The Docker containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it operates in isolation and runs its own instance of the software application without sharing resources or critical system elements with other containers. This ensures that each container has its own isolated context. Docker containers can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same or a different function.</p> <p><i>See, e.g.:</i></p> <p>Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see Deploying Docker containers on Amazon ECS.</p> <p>https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023</p> <p>A Docker image is a read-only template that defines your container. The image contains the code that will run including any definitions for any libraries and dependencies your code needs. A Docker container is an instantiated (running) Docker image. AWS provides Amazon Elastic Container Registry (ECR), an image registry for storing and quickly retrieving Docker images.</p> <p>https://aws.amazon.com/docker/#, Last accessed on June 14, 2023</p>

Claim 1	Accused Instrumentalities
	<p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image, Last accessed on June 14, 2023</p> <p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/, Last accessed on June 14, 2023</p>

Claim 1	Accused Instrumentalities
	<p>By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.</p> <p>https://docs.docker.com/get-started/overview/, Last accessed on June 14, 2023</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer structure of a container. At the top is a dashed box labeled 'Thin R/W layer' with an arrow pointing to it from the label 'Container layer'. Below this is a stack of four solid boxes representing image layers, labeled 'Image layers (R/O)' with a bracket and a padlock icon. The layers, from top to bottom, are: '91e54dfb1179' (0 B), 'd74508fb6632' (1.895 KB), 'c22013c84729' (194.5 KB), and 'd3a1f33e8a5a' (188.1 MB). The bottom layer is labeled 'ubuntu:15.04'. The entire stack is enclosed in a box labeled 'Container (based on ubuntu:15.04 image)'.</p> <p>https://docs.docker.com/storage/storagedriver/, Last accessed on June 14, 2023</p>

Claim 1	Accused Instrumentalities
	 <p data-bbox="653 906 1833 979">https://www.researchgate.net/figure/Docker-container-architecture_fig1_333235708, Last accessed on June 14, 2023</p>
<p data-bbox="247 1008 625 1373">[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a</p>	<p data-bbox="653 1008 1877 1138">In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p> <p data-bbox="653 1166 1877 1365">For example, In Docker, each container operates independently, and a Docker base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When a Docker image is used to create a container in ECS, an instance of the SLCSE is provided to a software application. Therefore, different instances of the SLCSE are</p>

Claim 1	Accused Instrumentalities
second instance of the SLCSE simultaneously.	<p>provided to different applications for performing either a same or a different function, simultaneously.</p> <p><i>See, e.g.:</i></p> <p>Amazon ECS uses Docker images in task definitions to launch containers. Docker is a technology that provides the tools for you to build, run, test, and deploy distributed applications in containers. Docker provides a walkthrough on deploying containers on Amazon ECS. For more information, see Deploying Docker containers on Amazon ECS.</p> <p>https://docs.aws.amazon.com/pdfs/AmazonECS/latest/developerguide/ecs-dg.pdf, Last accessed on June 14, 2023</p> <p>A Docker image is a read-only template that defines your container. The image contains the code that will run including any definitions for any libraries and dependencies your code needs. A Docker container is an instantiated (running) Docker image. AWS provides Amazon Elastic Container Registry (ECR), an image registry for storing and quickly retrieving Docker images.</p> <p>https://aws.amazon.com/docker/#, Last accessed on June 14, 2023</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image, Last accessed on June 14, 2023</p>

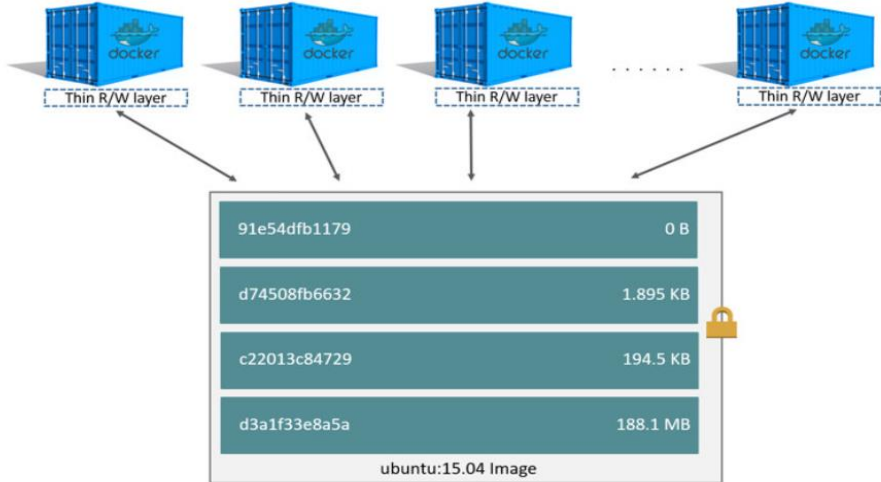
Claim 1	Accused Instrumentalities
	<p data-bbox="667 321 1556 451">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p data-bbox="653 995 1606 1027">https://docs.docker.com/storage/storagedriver/, Last accessed on June 14, 2023</p>

EXHIBIT E



Innovation, Science and
Economic Development Canada
Corporations Canada

Innovation, Sciences et
Développement économique Canada
Corporations Canada

Corporations Canada
C. D. Howe Building
235 Queen St
Ottawa ON K1A 0H5

Corporations Canada
Édifice C.D.Howe
235 rue Queen
Ottawa ON K1A 0H5

Corporate Profile / Profil corporatif

Date and time of Corporate Profile (YYYY-MM-DD)	2024-03-18 9:51 PM	(AAAA-MM-JJ) Date et heure du Profil corporatif
---	--------------------	---

CORPORATE INFORMATION	RENSEIGNEMENTS CORPORATIFS
Corporate name	Dénomination
	VirtaMove Corp.
Corporation number	761461-6
Business number	856544515RC0001
Governing legislation	Régime législatif
	Canada Business Corporations Act (CBCA) - 2010-08-12
	Loi canadienne sur les sociétés par actions (LCSA) - 2010-08-12
Status	Statut
	Active
	Active

REGISTERED OFFICE ADDRESS	ADRESSE DU SIÈGE
	110 Didsbury Road, M083 Ottawa ON K2T 0C2 Canada

ANNUAL FILINGS	DÉPÔTS ANNUELS
Anniversary date (MM-DD)	08-12 (MM-JJ) Date anniversaire
Filing period (MM-DD)	08-12 to/au 10-11 (MM-JJ) Période de dépôt
Status of annual filings	Statut des dépôts annuels
	Not due 2024 N'est pas dû
	Filed 2023 Déposé
	Filed 2022 Déposé
Date of last annual meeting (YYYY-MM-DD)	Not available / Pas disponible (AAAA-MM-JJ) Date de la dernière assemblée annuelle
Type	Type
	Non-distributing corporation with 50 or fewer shareholders
	Société n'ayant pas fait appel au public et comptant 50 actionnaires ou moins

DIRECTORS		ADMINISTRATEURS
Minimum number	1	Nombre minimal
Maximum number	10	Nombre maximal
Current number	4	Nombre actuel
Scott Munro	1950 University Circle, East Palo Alto CA 94303, United States	
Greg Dee	110 Ferrier Avenue, Toronto ON M4K 3H4, Canada	
Joseph Alsop	c/o Woodford Farm Trust, 255-800 South Street, Waltham MA 02453, United States	
Nigel Stokes	60 Boswell Avenue, Toronto ON M5R 1M4, Canada	

CORPORATE HISTORY		HISTORIQUE CORPORATIF
Corporate name history (YYYY-MM-DD)		(AAAA-MM-JJ) Historique de la dénomination
2010-08-12 to / à 2018-05-08	APPZERO SOFTWARE CORP.	
2018-05-08 to present / à maintenant	VirtaMove Corp.	
Certificates issued (YYYY-MM-DD)		(AAAA-MM-JJ) Certificats émis
Certificate of Incorporation	2010-08-12	Certificat de constitution en société
Certificate of Amendment	2014-02-18	Certificat de modification
Amendment details:		Renseignements concernant les modifications aux statuts :
Other		Autre
Certificate of Amendment	2015-02-18	Certificat de modification
Amendment details:		Renseignements concernant les modifications aux statuts :
Other		Autre
Certificate of Amendment	2016-02-12	Certificat de modification
Amendment details:		Renseignements concernant les modifications aux statuts :
Other		Autre
Certificate of Amendment	2018-05-08	Certificat de modification
Amendment details:		Renseignements concernant les modifications aux statuts :
Corporate name		Dénomination sociale
Amendments details are only available for amendments effected after 2010-03-20. Some certificates issued prior to 2000 may not be listed.		Seuls les renseignements concernant les modifications effectuées après 2010-03-20 sont disponibles. Certains certificats émis avant 2000 pourraient ne pas être listés.
Documents filed (YYYY-MM-DD)		(AAAA-MM-JJ) Documents déposés

The Corporate Profile sets out the most recent information filed with and accepted by Corporations Canada as of the date and time set out on the Profile.	Le Profil corporatif fait état des renseignements fournis et acceptés par Corporations Canada à la date et à l'heure indiquées dans le profil.
---	--

EXHIBIT F

8/5/24, 10:11 AM

Search | California Secretary of State

VIRTAMOVE, INC. (3552034)

Request
Certificate

<i>Initial Filing Date</i>	04/12/2013
<i>Status</i>	Active
<i>Standing - SOS</i>	Good
<i>Standing - FTB</i>	Good
<i>Standing - Agent</i>	Good
<i>Standing - VCFCF</i>	Good
<i>Formed In</i>	DELAWARE
<i>Entity Type</i>	Stock Corporation - Out of State - Stock
<i>Principal Address</i>	Canada 1 HINES ROAD, SUITE 204 OTTAWA ON, K2K 3C7
<i>Mailing Address</i>	Canada 1 HINES ROAD, SUITE 204 OTTAWA ON,K2K 3C7
<i>Statement of Info Due Date</i>	04/30/2023
<i>Agent</i>	1505 Corporation C T CORPORATION SYSTEM
<i>CA Registered Corporate (1505) Agent Authorized Employee(s)</i>	AMANDA GARCIA 330 N BRAND BLVD, GLENDALE, CA GABRIELA SANCHEZ 330 N BRAND BLVD, GLENDALE, CA DAISY MONTENEGRO 330 N BRAND BLVD, GLENDALE, CA BEATRICE CASAREZ- BARRIENTEZ 330 N BRAND BLVD, GLENDALE, CA JESSIE GASTELUM 330 N BRAND BLVD, GLENDALE, CA

8/5/24, 10:11 AM

Search | California Secretary of State

JOHN MONTIJO
330 N BRAND BLVD,
GLENDALE, CA

DIANA RUIZ
330 N BRAND BLVD,
GLENDALE, CA

SARAI MARIN
330 N BRAND BLVD,
GLENDALE, CA

EMANUEL JACOBO
330 N BRAND BLVD,
GLENDALE, CA

GLADYS AGUILERA
330 N BRAND BLVD,
GLENDALE, CA

VIVIAN IMPERIAL
330 N BRAND BLVD,
GLENDALE, CA

CARLOS PAZ
330 N BRAND BLVD,
GLENDALE, CA

ALBERTO DAMONTE
330 N BRAND BLVD,
GLENDALE, CA

PETER CAYETANO
330 N BRAND BLVD,
GLENDALE, CA

ELSA MONTANEZ
330 N BRAND BLVD,
GLENDALE, CA

XENIA PEREZ
330 N BRAND BLVD,
GLENDALE, CA

YESENIA CARPENTER
330 N BRAND BLVD,
GLENDALE, CA

JAQUELINE MEJIA
330 N BRAND BLVD,
GLENDALE, CA

[View History](#)[Request Access](#)

EXHIBIT G

**RED HAT
OPEN SOURCE ASSURANCE AGREEMENT**

PLEASE READ THESE TERMS CAREFULLY BEFORE AGREEING. IF YOU ARE ACTING ON BEHALF OF AN ENTITY, YOU REPRESENT THAT YOU HAVE THE AUTHORITY TO ENTER INTO THIS OPEN SOURCE ASSURANCE AGREEMENT ON BEHALF OF THE ENTITY. IF YOU DO NOT AGREE TO THESE TERMS, YOU MAY CONTINUE TO USE THE COVERED SOFTWARE TO THE EXTENT PERMITTED BY OTHER AGREEMENTS WITH RED HAT BUT YOU WILL NOT BE COVERED UNDER RED HAT'S OPEN SOURCE ASSURANCE PROGRAM AS PROVIDED HEREIN.

This Open Source Assurance Agreement ("**OSA Agreement**") is between Client and Red Hat, Inc., a Delaware corporation, with a principal place of business at 100 East Davie Street, Raleigh, North Carolina 27601, U.S.A. ("**Red Hat**"). The effective date of this OSA Agreement is the date that Client, or the authorized individual acting on its behalf, accepted this OSA Agreement ("**Effective Date**"). When we use a capitalized term without defining it in this OSA Agreement, the term has the meaning defined in the Red Hat Enterprise Agreement, Product Appendices and Order Form, if applicable (together, the "**Enterprise Agreement**"). "Client" or "you" means the person or entity acquiring the right to use or access the Red Hat Products and which is a party to this OSA Agreement.

You have obtained Red Hat Products that contain or run the Covered Software (defined below) from Red Hat or a Red Hat Business Partner under your Enterprise Agreement with Red Hat, and are interested in additional assurances for such Covered Software as described below.

By accepting this OSA Agreement, you agree to the following:

1. Open Source Assurance Program

If a third party asserts a legal claim against you alleging that your use of the Covered Software infringes or misappropriates the third party's intellectual property rights (a "**Claim**") and you have been and continue to be in compliance with the terms of the Enterprise Agreement, then Red Hat will (a) defend you against the Claim and (b) pay costs, damages and attorneys' fees that are attributable to your use of the Covered Software and included in a final judgment against you or in a final settlement approved by Red Hat.

Red Hat may, at any time, at its sole expense and option: (i) obtain the rights necessary for you to continue to use the Covered Software; (ii) modify the Covered Software; or (iii) provide code of similar functionality to replace the portion of the Covered Software that is the subject of the Claim (collectively (i), (ii) and (iii) are the "**IP Resolutions**"); provided that, if none of the IP Resolutions are available on a basis that Red Hat finds commercially reasonable, then Red Hat may terminate subscriptions or access to the Red Hat Product that is the subject of the Claim without further liability under this OSA Agreement for your further use of the Covered Software, and, if you return the Covered Software that is subject to the Claim, Red Hat will refund pro-rata any prepaid fees with respect to the Red Hat Product with the Covered Software that is subject to the Claim.

As conditions precedent to Red Hat's obligations under this OSA Agreement, you must: (i) be current in the payment of all applicable fees at the time of the Claim; (ii) notify Red Hat in writing no later than ten (10) days following receipt of any Claim unless Red Hat is not prejudiced by any delay; (iii) grant Red Hat the right to control and conduct the defense of the Claim with counsel of Red Hat's choice and settle such Claim at Red Hat's sole discretion; and (iv) cooperate with Red Hat in the management and/or defense of the Claim.

Red Hat will have no obligations with regard to any Claim that is based on: (i) a modification of Covered Software not made by or at the written direction of Red Hat; (ii) Red Hat's compliance with designs, specifications or instructions provided by you; (iii) use of the Covered Software in combination with a product, process, step, structure, data, or business method not provided by Red Hat, if the infringement or misappropriation would not have occurred without the combined use; (iv) facts or circumstances constituting a breach of the Enterprise Agreement by you; (v) use of any release of the Covered Software if the infringement or misappropriation identified in the Claim would not have occurred or would have been less likely through use of a more recent release of the Covered Software or after notice by Red Hat (whether before or during a Claim) to discontinue use of all or a portion of the Covered Software; or (vi) any use of the Covered Software by you other than for your own internal use. If the Claim includes allegation(s) against you regarding matters other than the Covered Software, you are responsible for the proportional share of fees and costs in defending such allegations and Red Hat will only be responsible for the proportional share of fees and costs attributable to the Covered Software.

2. Covered Software

"**Covered Software**" is Red Hat branded software that is included in a Red Hat Product offering for which you have an active paid subscription at the time of a Claim. Covered Software does not include: (i) third party products or offerings; (ii) products, packages or code not provided or supported as part of a Red Hat Product, such as Fedora, Centos Stream or WildFly.org; (iii) software or services provided at no charge or for evaluation, preview or demonstration purposes; or (iv) any programs listed or described at <https://www.redhat.com/en/about/third-party-end-user-agreements>.

3. Limitations on Liability

Red Hat will not be obligated to defend or pay any amounts in connection with a Claim related to any period of time during which you do not have active, paid subscriptions to Red Hat Products with Covered Software that is the subject of the Claim. Red Hat will have no obligation to you under this OSA Agreement if, as of the Effective Date, you have received notice of allegations of infringement or misappropriation, are engaged in litigation, or have received an invitation to license, in each case concerning the subject matter of what would otherwise be a Claim under this OSA Agreement or with respect to a product substantially similar to the Covered Software.

NOTWITHSTANDING ANYTHING TO THE CONTRARY, RED HAT'S AGGREGATE AND CUMULATIVE LIABILITY UNDER BOTH THIS OSA AGREEMENT AND THE ENTERPRISE AGREEMENT SHALL BE SUBJECT TO THE LIMITATIONS OF LIABILITY CONTAINED IN THE ENTERPRISE AGREEMENT IN EFFECT AS OF THE DATE OF A CLAIM; PROVIDED THAT IN NO EVENT WILL RED HAT'S AND ITS AFFILIATES' AGGREGATE AND CUMULATIVE LIABILITY TO YOU EXCEED THE TOTAL FEES PAID TO RED HAT WITH RESPECT TO YOUR PURCHASES OF RED HAT PRODUCTS (DIRECTLY OR INDIRECTLY FROM A RED HAT BUSINESS PARTNER) DURING THE 12 MONTH PERIOD IMMEDIATELY PRECEDING YOUR NOTICE TO RED HAT OF THE CLAIM. NOTWITHSTANDING ANYTHING TO THE CONTRARY CONTAINED IN THIS OSA AGREEMENT OR THE ENTERPRISE AGREEMENT, IN NO EVENT WILL RED HAT OR ITS AFFILIATES BE LIABLE TO YOU OR YOUR AFFILIATES FOR ANY INCIDENTAL,

CONSEQUENTIAL, SPECIAL, INDIRECT, EXEMPLARY OR PUNITIVE DAMAGES, WHETHER ARISING IN TORT, CONTRACT, OR OTHERWISE; OR FOR ANY DAMAGES ARISING OUT OF OR IN CONNECTION WITH ANY MALFUNCTIONS, DELAYS, LOSS OF DATA, LOST PROFITS, LOST SAVINGS, INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS, EVEN IF RED HAT OR ITS AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES AND EVEN IF SUCH DAMAGES WERE FORESEEABLE.

This OSA Agreement sets forth your exclusive remedies and Red Hat's sole obligations for claims arising from or related to intellectual property rights and supersedes any other Red Hat obligation related to the subject matter. If any other applicable indemnity coverage or remedy is available to you related to intellectual property infringement, you agree that the total of all benefits payable under all such provisions will not exceed the total damages, costs, and expenses incurred by you, and that Red Hat will pay only its proportional share, subject to the limitations in this Section 3. Red Hat will have no obligations under this OSA Agreement with respect to any legal action that does not or no longer includes a Claim for which Red Hat is responsible under this OSA Agreement.

4. Term and Survival

The term of this OSA Agreement will begin on the Effective Date and will continue for such period as you have active, paid Red Hat subscriptions for Red Hat Products. If Red Hat updates or amends its Open Source Assurance program, (i) this OSA Agreement will apply only until the end of the then current subscription period (or one month later for on-demand subscriptions) and (ii) you will have the opportunity, if you so elect, to participate in the updated or amended OSA Agreement for any additional subscriptions or renewal that are purchased on or after the date Red Hat updates or amends the Open Source Assurance program. If this OSA Agreement is terminated for any reason, Sections 2 - 6 will survive termination.

5. Governing Law

This OSA Agreement, and any claim, controversy or dispute related to the OSA Agreement, are governed by and construed in accordance with the governing laws of the state of New York without giving effect to any conflicts of laws provisions. To the extent permissible, the United Nations Convention on Contracts for the International Sale of Goods will not apply, even if adopted. To the fullest extent permitted, each party waives the right to trial by jury in any legal proceeding arising out of or relating to this OSA Agreement.

6. Miscellaneous

(i) Notices must be in English, in writing, and will be deemed given upon receipt after being sent using a method that provides for positive confirmation of delivery, including through an automated receipt or by electronic log, to the address or email address indicated on your Red Hat account registration form or to Red Hat at: Red Hat, Inc., Attention: General Counsel, 100 East Davie Street, Raleigh, North Carolina 27601, U.S.A. or via email: legal-notices@redhat.com. (ii) This OSA Agreement is binding on the parties to this OSA Agreement, and nothing in this OSA Agreement grants any other person or entity any right, benefit or remedy. (iii) This OSA Agreement is assignable by either party only with the other party's prior written consent; provided that Red Hat may assign this OSA Agreement to an affiliate or pursuant to a merger or a sale of all or substantially all of its assets or stock without the prior approval. (iv) The delay or failure of Red Hat to exercise any rights under this OSA Agreement will not constitute a waiver or forfeiture of such rights. A waiver by a party under this OSA Agreement is only valid if in writing and signed by an authorized representative of the party. If any provision of this OSA Agreement is held invalid or unenforceable for any reason, this OSA Agreement will be deemed invalid in its entirety.

EXHIBIT H



Overview of images

- [Understanding containers, images, and image streams](#)
- [Images](#)
- [Image registry](#)
- [Image repository](#)
- [Image tags](#)
- [Image IDs](#)
- [Containers](#)
- [Why use imagestreams](#)
- [Image stream tags](#)
- [Image stream images](#)
- [Image stream triggers](#)
- [How you can use the Cluster Samples Operator](#)
- [About templates](#)
- [How you can use Ruby on Rails](#)

Understanding containers, images, and image streams

Containers, images, and image streams are important concepts to understand when you set out to create and manage containerized software. An image holds a set of software that is ready to run, while a container is a running instance of a container image. An image stream provides a way of storing different versions of the same basic image. Those different versions are represented by different tags on the same image name.

Images

Containers in OpenShift Container Platform are based on OCI- or Docker-formatted container *images*. An image is a binary that includes all of the requirements for running a single container, as well as metadata describing its needs and capabilities.

You can think of it as a packaging technology. Containers only have access to resources defined in the image unless you give the container additional access when creating it. By deploying the same image in multiple containers across multiple hosts and load balancing between them, OpenShift Container Platform can provide redundancy and horizontal scaling for a service packaged into an image.

You can use the `podman` or `docker` CLI directly to build images, but OpenShift Container Platform also supplies builder images that assist with creating new images by adding your code or configuration to existing images.

Because applications develop over time, a single image name can actually refer to many different versions of the same image. Each different image is referred to uniquely by its hash, a long hexadecimal number such as `fd44297e2ddb050ec4f...`, which is usually shortened to 12 characters, such as `fd44297e2ddb`.

You can create, manage, and use container images.

Image registry

An image registry is a content server that can store and serve container images. For example:

```
registry.redhat.io
```

A registry contains a collection of one or more image repositories, which contain one or more tagged images. Red Hat provides a registry at `registry.redhat.io` for subscribers. OpenShift Container Platform can also supply its own OpenShift image registry for managing custom container images.

Image repository

An image repository is a collection of related container images and tags identifying them. For example, the OpenShift Container Platform Jenkins images are in the repository:

```
docker.io/openshift/jenkins-2-centos7
```

Image tags

An image tag is a label applied to a container image in a repository that distinguishes a specific image from other images in an image stream. Typically, the tag represents a version number of some sort. For example, here `:v3.11.59-2` is the tag:

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

You can add additional tags to an image. For example, an image might be assigned the tags `:v3.11.59-2` and `:latest`.

OpenShift Container Platform provides the `oc tag` command, which is similar to the `docker tag` command, but operates on image streams instead of directly on images.

Image IDs

An image ID is a SHA (Secure Hash Algorithm) code that can be used to pull an image. A SHA image ID cannot change. A specific SHA identifier always references the exact same container image content.

For example:

```
docker.io/openshift/jenkins-2-centos7@sha256:ab312bda324
```

Containers

The basic units of OpenShift Container Platform applications are called containers. [Linux container technologies](#) are lightweight mechanisms for isolating running processes so that they are limited to interacting with only their designated resources. The word container is defined as a specific running or paused instance of a container image.

Many application instances can be running in containers on a single host without visibility into each others' processes, files, network, and so on. Typically, each container provides a single service, often called a micro-service, such as a web server or a database, though containers can be used for arbitrary workloads.

The Linux kernel has been incorporating capabilities for container technologies for years. The Docker project developed a convenient management interface for Linux containers on a host. More recently, the [Open Container Initiative](#) has developed open standards for container formats and container runtimes. OpenShift Container Platform and Kubernetes add the ability to orchestrate OCI- and Docker-formatted containers across multi-host installations.

Though you do not directly interact with container runtimes when using OpenShift Container Platform, understanding their capabilities and terminology is important for understanding their role in OpenShift Container Platform and how your applications function inside of containers.

Tools such as [podman](#) can be used to replace `docker` command-line tools for running and managing containers directly. Using `podman`, you can experiment with containers separately from OpenShift Container Platform.

Why use imagestreams

An image stream and its associated tags provide an abstraction for referencing container images from within OpenShift Container Platform. The image stream and its tags allow you to see what images are available and ensure that you are using the specific image you need even if the image in the repository changes.

Image streams do not contain actual image data, but present a single virtual view of related images, similar to an image repository.

You can configure builds and deployments to watch an image stream for notifications when new images are added and react by performing a build or deployment, respectively.

For example, if a deployment is using a certain image and a new version of that image is created, a deployment could be automatically performed to pick up the new version of the image.

However, if the image stream tag used by the deployment or build is not updated, then even if the container image in the container image registry is updated, the build or deployment continues using the previous, presumably known good image.

The source images can be stored in any of the following:

- OpenShift Container Platform's integrated registry.
- An external registry, for example registry.redhat.io or quay.io.
- Other image streams in the OpenShift Container Platform cluster.

When you define an object that references an image stream tag, such as a build or deployment configuration, you point to an image stream tag and not the repository. When you build or deploy your application, OpenShift Container Platform queries the repository using the image stream tag to locate the associated ID of the image and uses that exact image.

The image stream metadata is stored in the etcd instance along with other cluster information.

Using image streams has several significant benefits:

- You can tag, rollback a tag, and quickly deal with images, without having to re-push using the command line.
- You can trigger builds and deployments when a new image is pushed to the registry. Also, OpenShift Container Platform has generic triggers for other resources, such as Kubernetes objects.
- You can mark a tag for periodic re-import. If the source image has changed, that change is picked up and reflected in the image stream, which triggers the build or deployment flow, depending upon the build or deployment configuration.
- You can share images using fine-grained access control and quickly distribute images across your teams.
- If the source image changes, the image stream tag still points to a known-good version of the image, ensuring that your application does not break unexpectedly.
- You can configure security around who can view and use the images through permissions on the image stream objects.

- Users that lack permission to read or list images on the cluster level can still retrieve the images tagged in a project using image streams.

You can [manage](#) image streams, [use image streams with Kubernetes resources](#), and [trigger updates on image stream updates](#).

Image stream tags

An image stream tag is a named pointer to an image in an image stream. An image stream tag is similar to a container image tag.

Image stream images

An image stream image allows you to retrieve a specific container image from a particular image stream where it is tagged. An image stream image is an API resource object that pulls together some metadata about a particular image SHA identifier.

Image stream triggers

An image stream trigger causes a specific action when an image stream tag changes. For example, importing can cause the value of the tag to change, which causes a trigger to fire when there are deployments, builds, or other resources listening for those.

How you can use the Cluster Samples Operator

During the initial startup, the Operator creates the default samples resource to initiate the creation of the image streams and templates. You can use the Cluster Samples Operator to manage the sample image streams and templates stored in the `openshift` namespace.

As a cluster administrator, you can use the Cluster Samples Operator to:

- [Configure the Operator](#).
- [Use the Operator with an alternate registry](#).

About templates

A template is a definition of an object to be replicated. You can use [templates](#) to build and deploy configurations.

How you can use Ruby on Rails

As a developer, you can use [Ruby on Rails](#) to:

- Write your application:
 - Set up a database.
 - Create a welcome page.
 - Configure your application for OpenShift Container Platform.
 - Store your application in Git.
- Deploy your application in OpenShift Container Platform:
 - Create the database service.
 - Create the frontend service.
 - Create a route for your application.

EXHIBIT I

**UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

VIRTAMOVE, CORP.,

Plaintiff,

v.

INTERNATIONAL BUSINESS MACHINES
CORP.,

Defendant.

Case No.

JURY TRIAL DEMANDED

**COMPLAINT FOR PATENT INFRINGEMENT AGAINST
INTERNATIONAL BUSINESS MACHINES CORP.**

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, “Plaintiff” or “VirtaMove”) makes the following allegations against Defendant International Business Machines Corp. (collectively, “Defendant” or “IBM”):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant’s unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058 (collectively, the “Asserted Patents”). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as Appzero Software Corp. (“Appzero”), which was established in

2010.

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repackage, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove's mission is to move and modernize the world's server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach \$2.1 billion in 2019 and \$4.3 billion in 2022—a compound annual growth rate (“CAGR”) of 30%. *See, e.g.,* <https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022>. Another source reported the application containers market had a market size of \$5.45 billion in 2024 and estimated it to reach \$19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.,* <https://www.mordorintelligence.com/industry-reports/application-container-market>.

6. Defendant International Business Machines Corp. is a New York corporation. IBM has a principal place of business at One New Orchard Road, Armonk, New York 10504. IBM may also be served with process via its registered agent Amanda Garcia, at 330 North Brand Blvd,

Glendale, California 91203.

JURISDICTION AND VENUE

7. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

9. Venue is proper in this District. For example, IBM has a regular and established place of business, including, e.g., at 1700 Summit Avenue, Plano, Texas 75074.

COUNT I

INFRINGEMENT OF U.S. PATENT NO. 7,519,814

10. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

11. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 ('814 patent), titled "System for Containerization of Application Sets," issued on April 14, 2009. A true and correct copy of the '814 Patent is attached as Exhibit 1.

12. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., IBM's IBM Cloud Kubernetes

Service, that directly infringe, literally and/or under the doctrine of equivalents, claims of the '814 patent, for example:

Run Kubernetes at enterprise scale

Experience a certified, managed Kubernetes solution, built for creating a cluster of compute hosts to deploy and manage containerized apps on IBM Cloud®. IBM manages the master, freeing you from having to administer the host OS, container runtime and Kubernetes version-update process.

<https://www.ibm.com/products/kubernetes-service>.

13. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

14. Defendant's infringement has been and is willful. Defendant knew of VirtaMove, its products, and at least one of the patents long before this suit was filed and at least as early as 2012. For example, on or about February 2012, in connection with the prosecution of U.S. Patent App. No. 12/146,322 (assigned to IBM), the examiner cited U.S. Pub. No. 2005/0060722 (which issued as the '814 Patent) against IBM. On or about November 26, 2012, in connection with the prosecution of U.S. Patent No. 8,893,306 (assigned to IBM), the examiner cited the '814 Patent against IBM. On or about June 2015, in connection with the prosecution of U.S. Patent No. 9,166,865 (assigned to IBM), the examiner cited U.S. Pub. No. 2005/0060722 against IBM. Defendant knew, or should have known, that its conduct amounted to infringement of the '814

patent. Accordingly, Defendant is liable for willful infringement.

15. Defendant also knowingly and intentionally induces infringement of claims of the '814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and/or earlier, as set forth above. Despite this knowledge of the '814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '814 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '814 patent, thereby specifically intending for and inducing its customers to infringe the '814 patent through the customers' normal and customary use of the Accused Products.

16. Defendant has also infringed, and continue to infringe, claims of the '814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '814 patent, in violation of 35 U.S.C. § 271(c).

17. The Accused Products satisfy all claim limitations of claims of the '814 patent. A claim chart comparing an independent claim of the '814 patent to a representative Accused Product, is attached as Exhibit 2, which is hereby incorporated by reference in its entirety.

18. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the '814 patent pursuant to 35 U.S.C. § 271.

19. As a result of Defendant's infringement of the '814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court.

COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

20. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

21. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 ('058 patent), titled "Computing System Having User Mode Critical System Elements as Shared Libraries," issued on August 24, 2010. A true and correct copy of the '058 patent is attached as Exhibit 3.

22. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., IBM's IBM Cloud Kubernetes Service, that directly infringe, literally and/or under the doctrine of equivalents, claims of the '058 patent, for example:

Run Kubernetes at enterprise scale

Experience a certified, managed Kubernetes solution, built for creating a cluster of compute hosts to deploy and manage containerized apps on IBM Cloud®. IBM manages the master, freeing you from having to administer the host OS, container runtime and Kubernetes version-update process.

<https://www.ibm.com/products/kubernetes-service>.

23. The infringement of the Asserted Patents is also attributable to Defendant. Defendant and/or users of the Accused Products directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

24. Defendant's infringement has been and is willful. Defendant knew of VirtaMove, its products, and at least one of the patents long before this suit was filed and at least as early as 2015. For example, U.S. Patent No. 9,176,713 (assigned to IBM) issued on November 3, 2015 and identifies the '058 Patent under "References Cited." Additionally, U.S. Patent No. 9,934,055 (assigned to IBM) issued on April 3, 2018 and identifies the '058 Patent under "References Cited." Defendant knew, or should have known, that its conduct amounted to infringement of the '058 patent. Accordingly, Defendant is liable for willful infringement.

25. Defendant also knowingly and intentionally induces infringement of claims of the '058 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed. Despite this knowledge of the '058 patent, Defendant continues to actively encourage and

instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '058 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '058 patent, thereby specifically intending for and inducing its customers to infringe the '058 patent through the customers' normal and customary use of the Accused Products.

26. Defendant has also infringed, and continue to infringe, claims of the '058 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '058 patent, in violation of 35 U.S.C. § 271(c).

27. The Accused Products satisfy all claim limitations of claims of the '058 patent. A claim chart comparing an independent claim of the '058 patent to a representative Accused Product, is attached as Exhibit 4, which is hereby incorporated by reference in its entirety.

28. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the '058 patent pursuant to 35 U.S.C. § 271.

29. As a result of Defendant's infringement of the '058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no

event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

- a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;
- b. A judgment in favor of Plaintiff that Defendant has willfully infringed the '814 and '058 patents;
- c. A permanent injunction prohibiting Defendant from further acts of infringement of each of the '814 and '058 patents;
- d. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant's infringement of each of the Asserted Patents;
- e. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;
- f. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys' fees against Defendant; and
- g. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury

of any issues so triable by right.

Dated: January 31, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)

rmirzaie@raklaw.com

Marc A. Fenster (CA SBN 181067)

mfenster@raklaw.com

Neil A. Rubin (CA SBN 250761)

nrubin@raklaw.com

Amy E. Hayden (CA SBN 287026)

ahayden@raklaw.com

Christian W. Conkle (CA SBN 306374)

cconkle@raklaw.com

Jonathan Ma (CA SBN 312773)

jma@raklaw.com

RUSS AUGUST & KABAT

12424 Wilshire Boulevard, 12th Floor

Los Angeles, CA 90025

Telephone: (310) 826-7474

Attorneys for Plaintiff VirtaMove, Corp.

Exhibit 1



US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,519,814 B2**
(45) **Date of Patent:** **Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

WO WO 2006/039181 A 4/2006

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248;
709/214; 719/319

(58) **Field of Classification Search** 713/167;
719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

OTHER PUBLICATIONS

Soltész, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667.*

Primary Examiner—Kambiz Zand

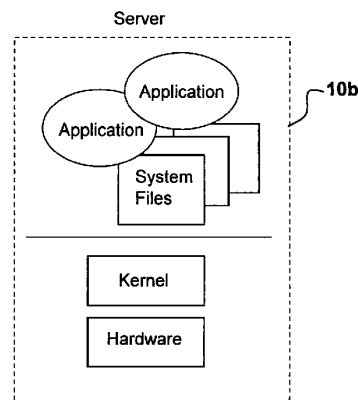
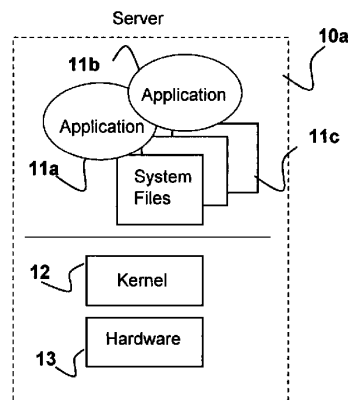
Assistant Examiner—Ronald Baum

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



US 7,519,814 B2

Page 2

U.S. PATENT DOCUMENTS				2002/0004854 A1	1/2002	Hartley	
6,847,970 B2 *	1/2005	Keller et al.	707/100	2002/0174215 A1	11/2002	Schaefer	709/224
7,076,784 B1 *	7/2006	Russell et al.	719/315	2003/0101292 A1	5/2003	Fisher	
7,287,259 B2 *	10/2007	Grier et al.	719/331	* cited by examiner			

U.S. Patent

Apr. 14, 2009

Sheet 1 of 17

US 7,519,814 B2

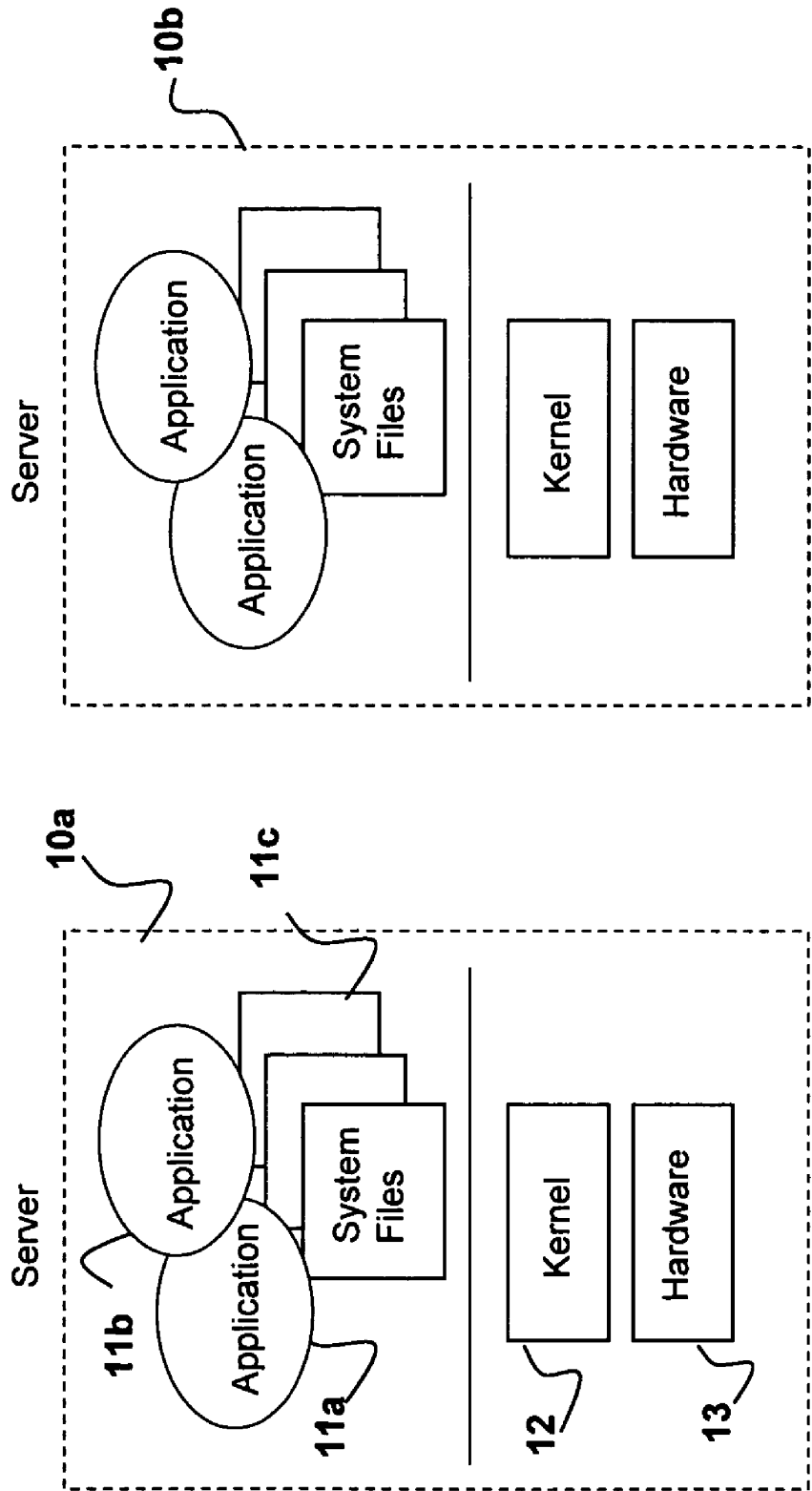


Figure 1

U.S. Patent

Apr. 14, 2009

Sheet 2 of 17

US 7,519,814 B2

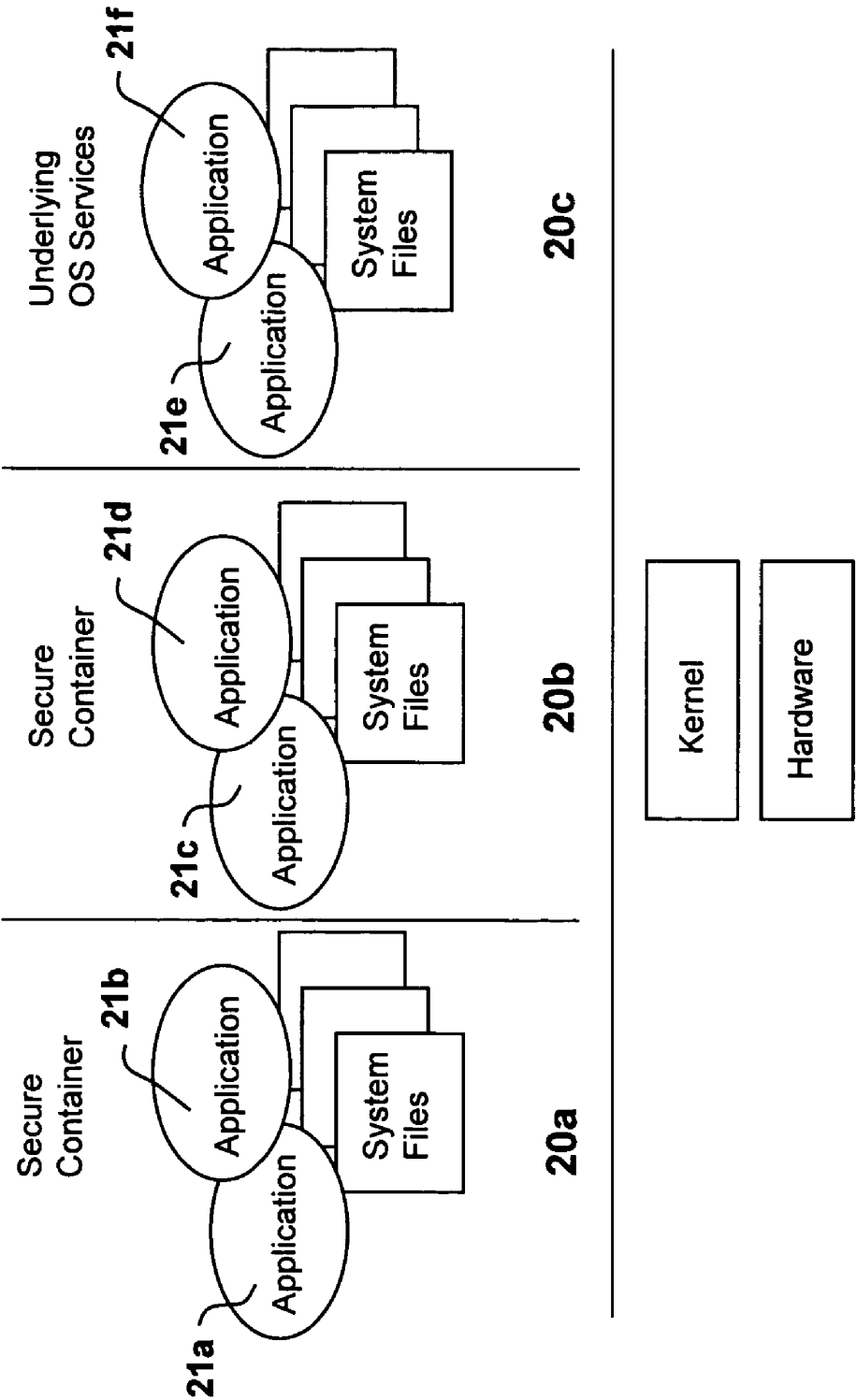


Figure 2

U.S. Patent

Apr. 14, 2009

Sheet 3 of 17

US 7,519,814 B2

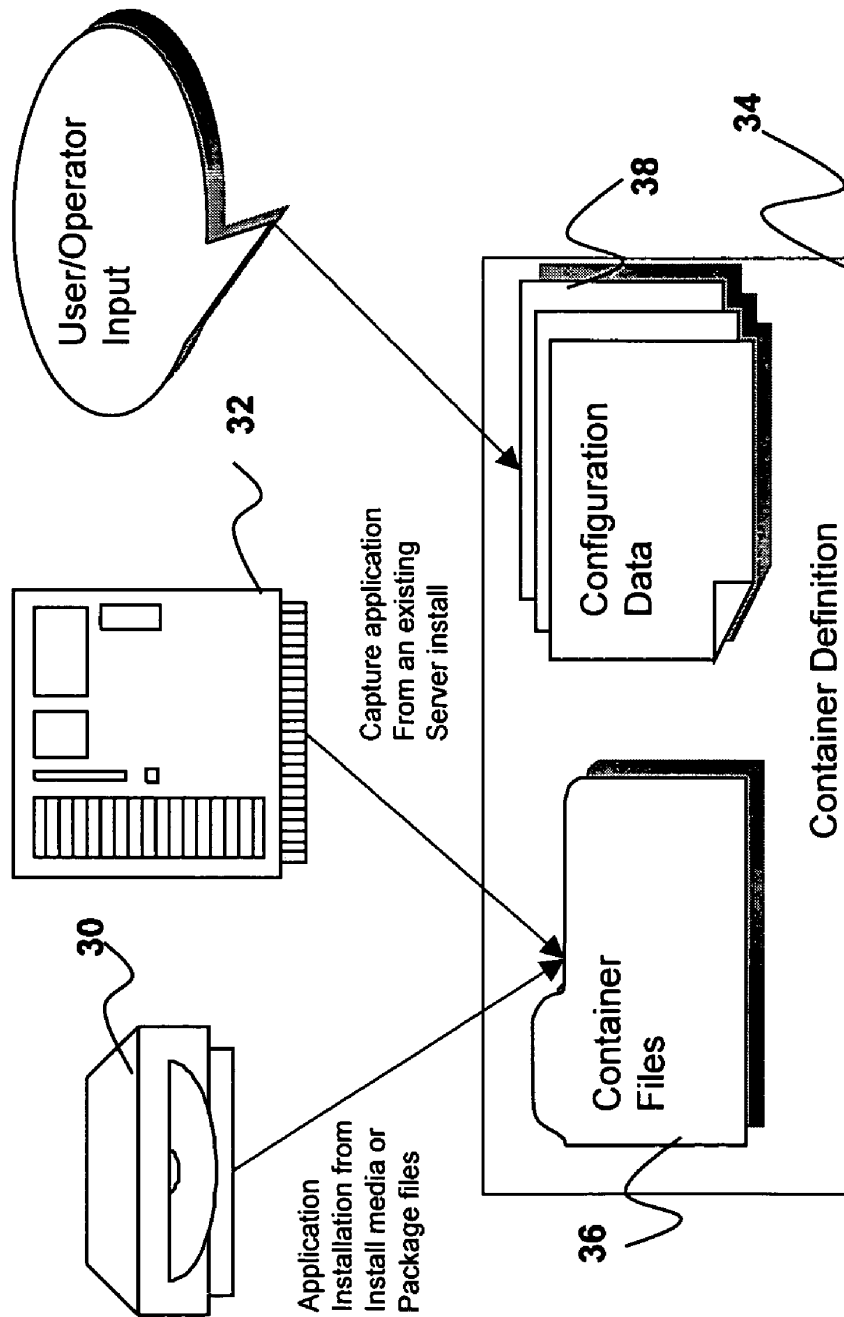


Figure 3

U.S. Patent

Apr. 14, 2009

Sheet 4 of 17

US 7,519,814 B2

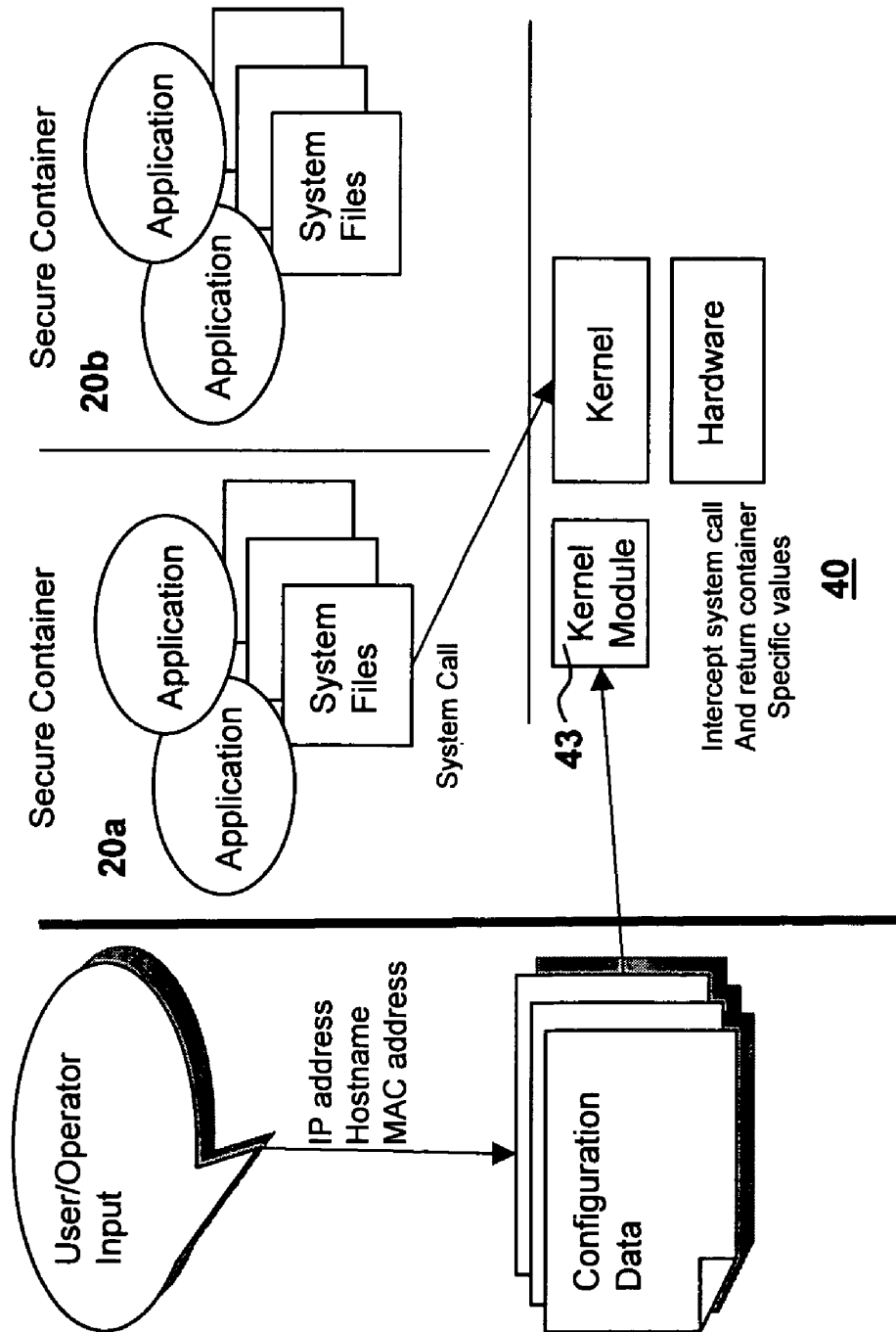


Figure 4

U.S. Patent

Apr. 14, 2009

Sheet 5 of 17

US 7,519,814 B2

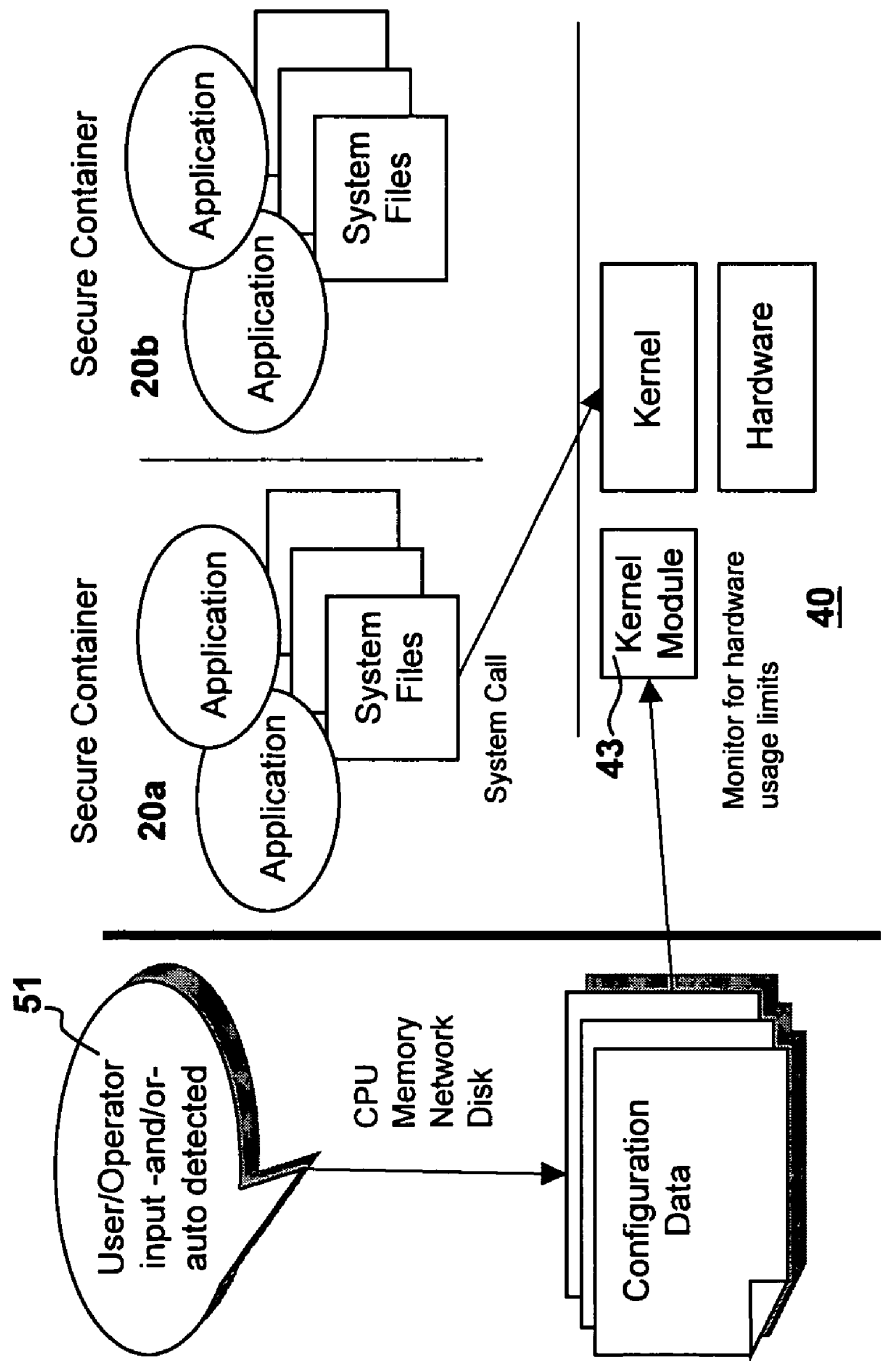


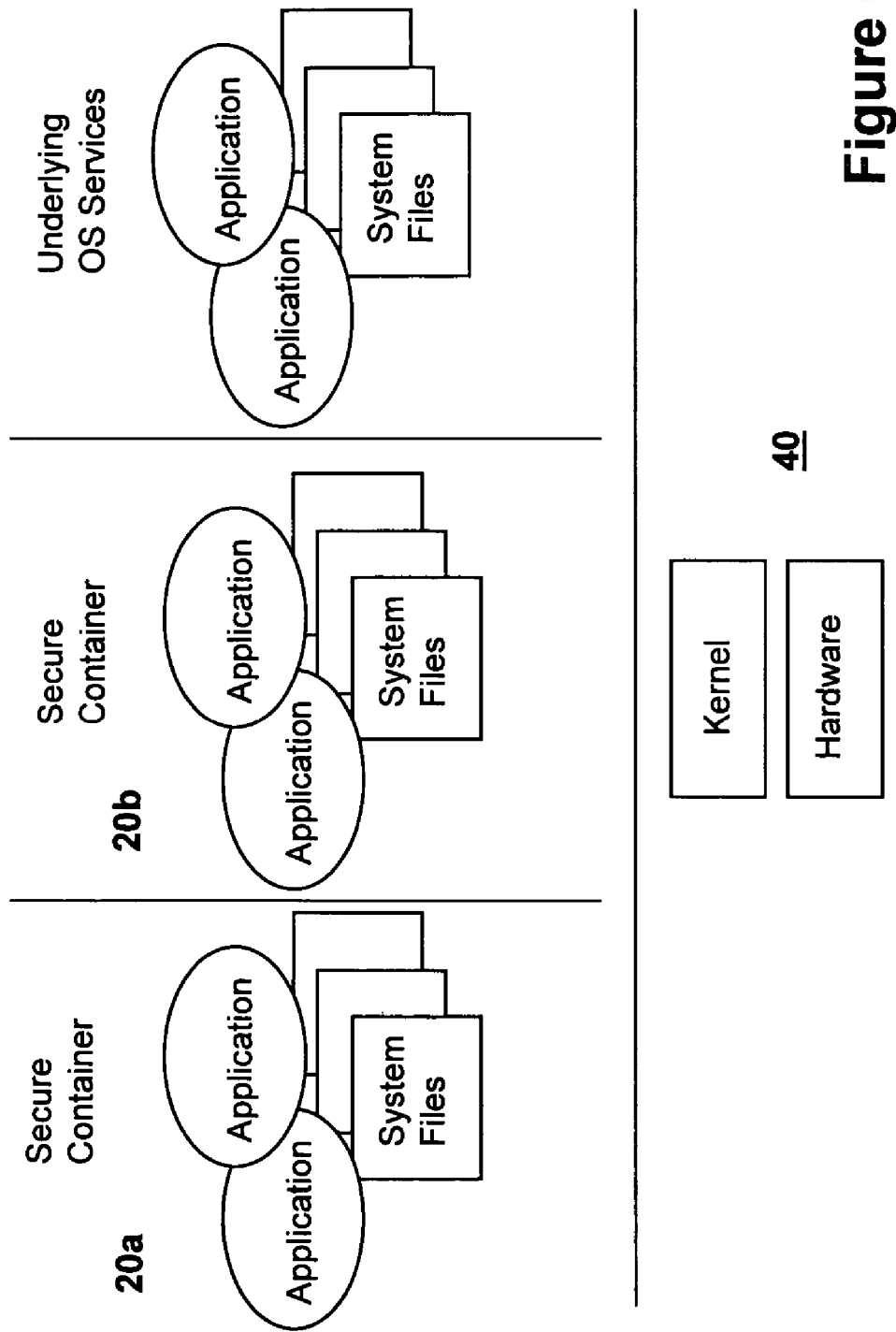
Figure 5

U.S. Patent

Apr. 14, 2009

Sheet 6 of 17

US 7,519,814 B2



U.S. Patent

Apr. 14, 2009

Sheet 7 of 17

US 7,519,814 B2

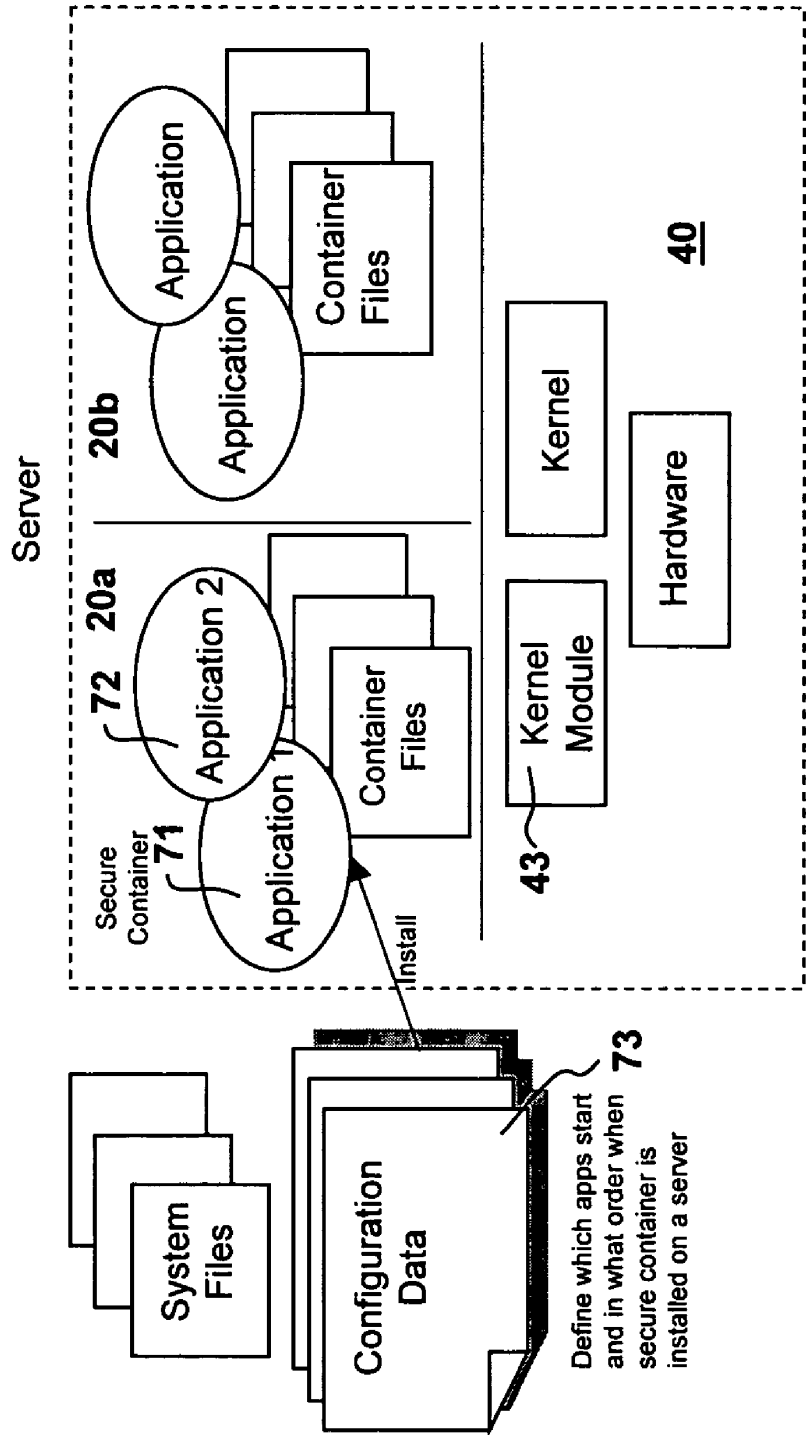


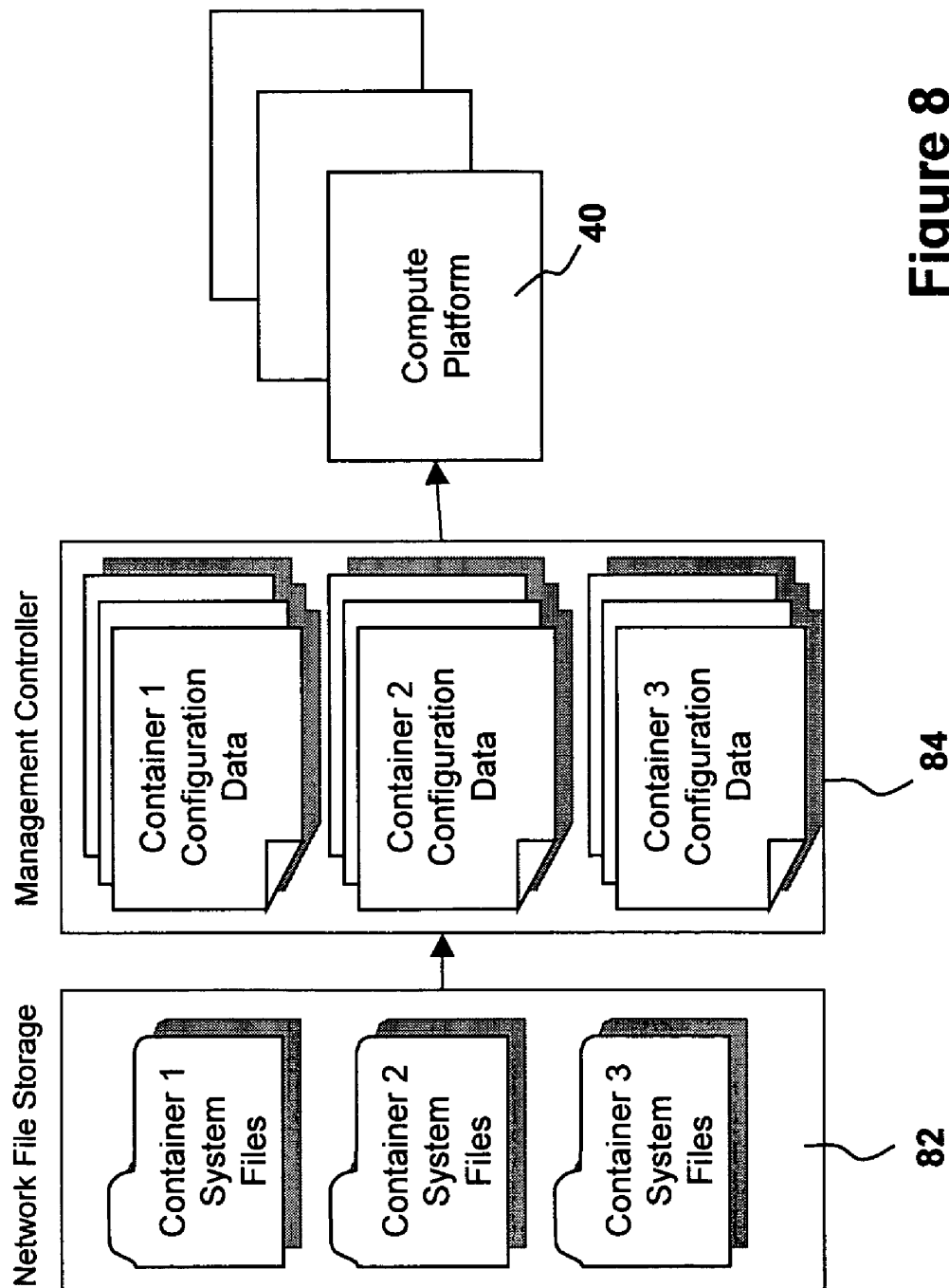
Figure 7

U.S. Patent

Apr. 14, 2009

Sheet 8 of 17

US 7,519,814 B2



U.S. Patent

Apr. 14, 2009

Sheet 9 of 17

US 7,519,814 B2

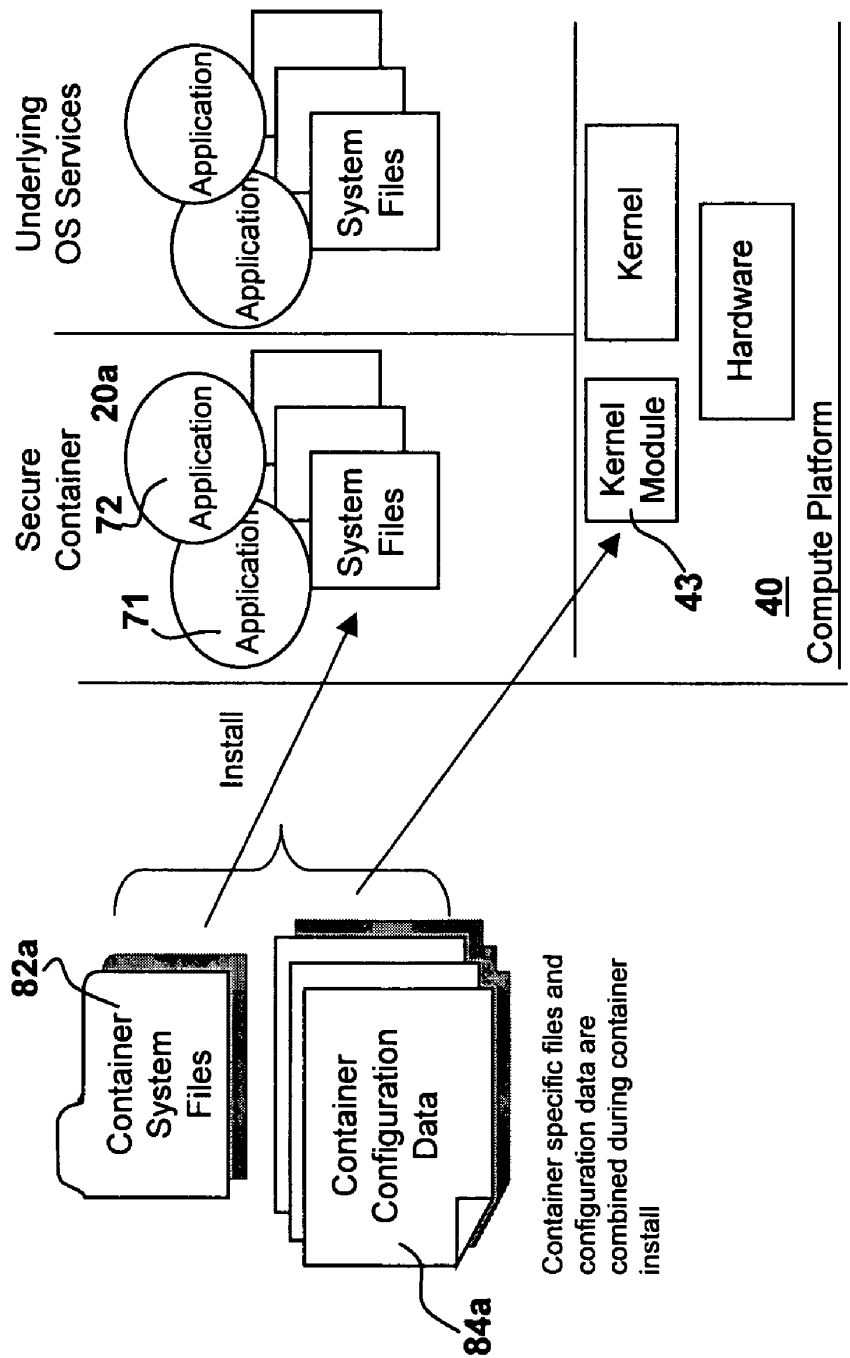


Figure 9

U.S. Patent

Apr. 14, 2009

Sheet 10 of 17

US 7,519,814 B2

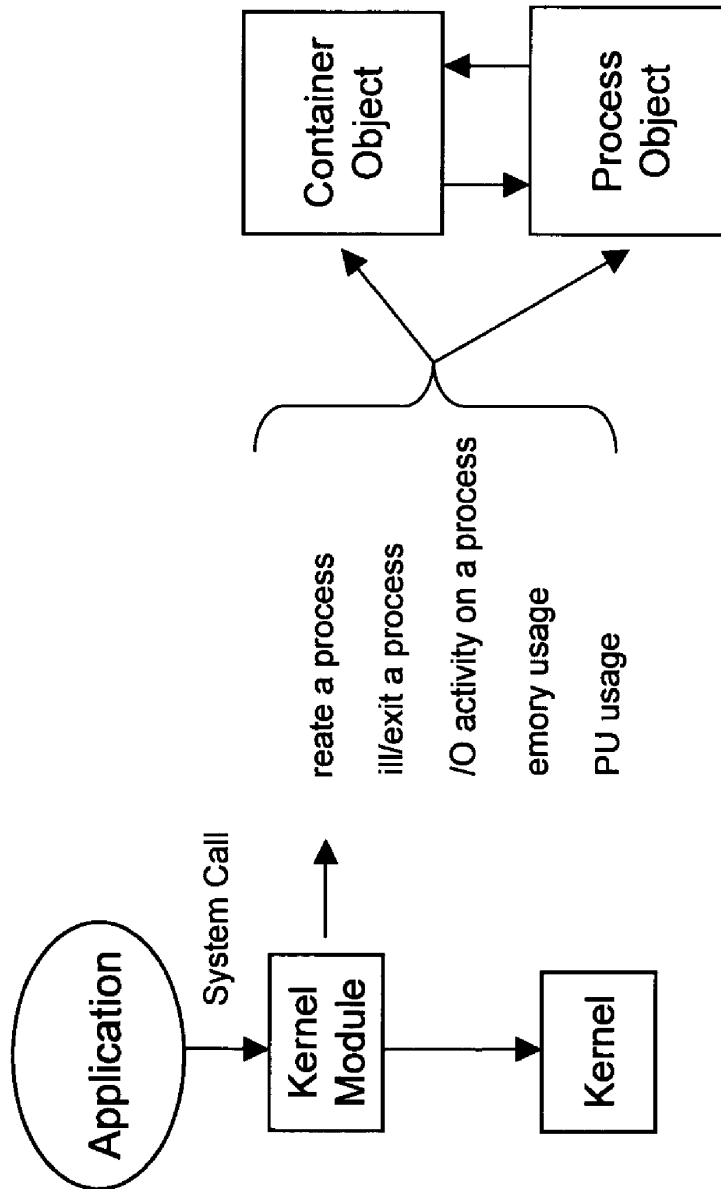


Figure 10

U.S. Patent

Apr. 14, 2009

Sheet 11 of 17

US 7,519,814 B2

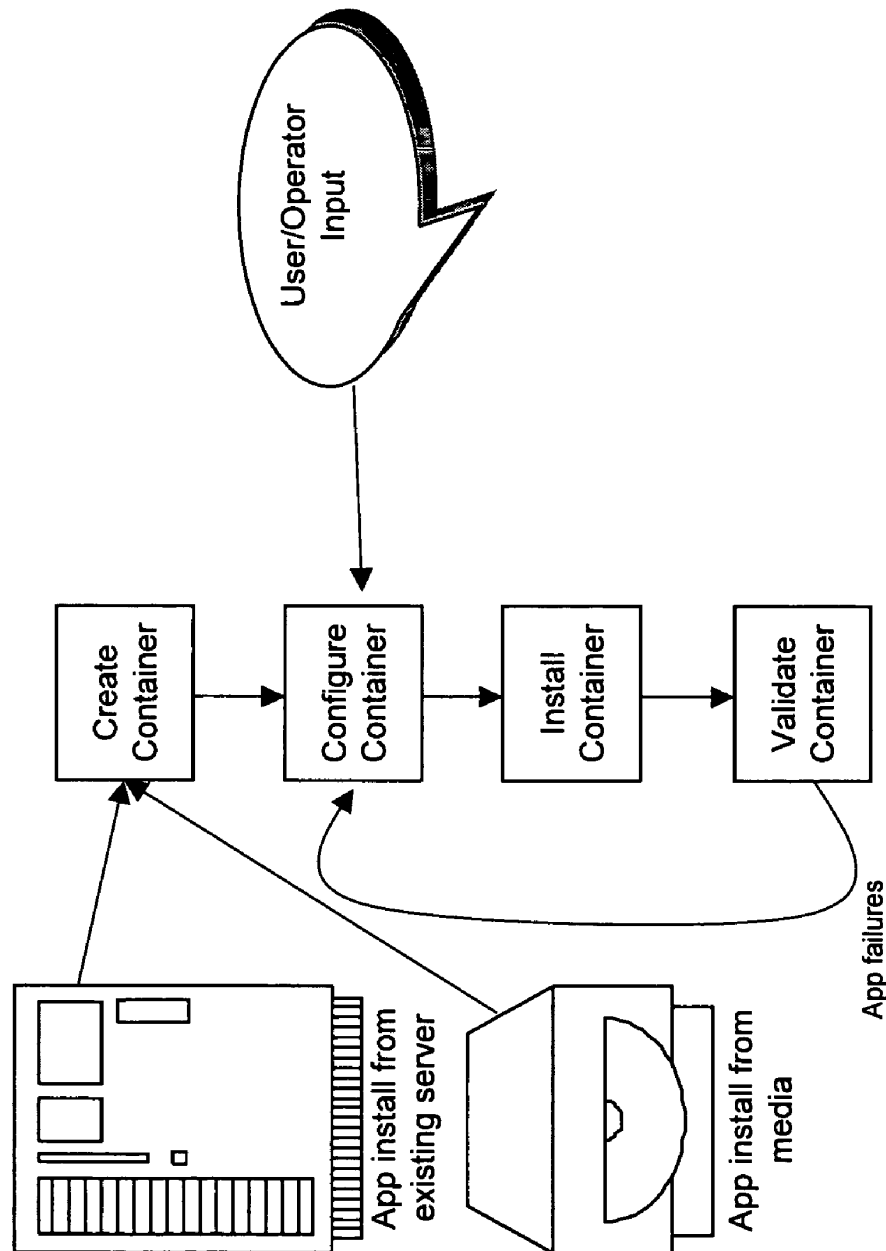


Figure 11

U.S. Patent

Apr. 14, 2009

Sheet 12 of 17

US 7,519,814 B2

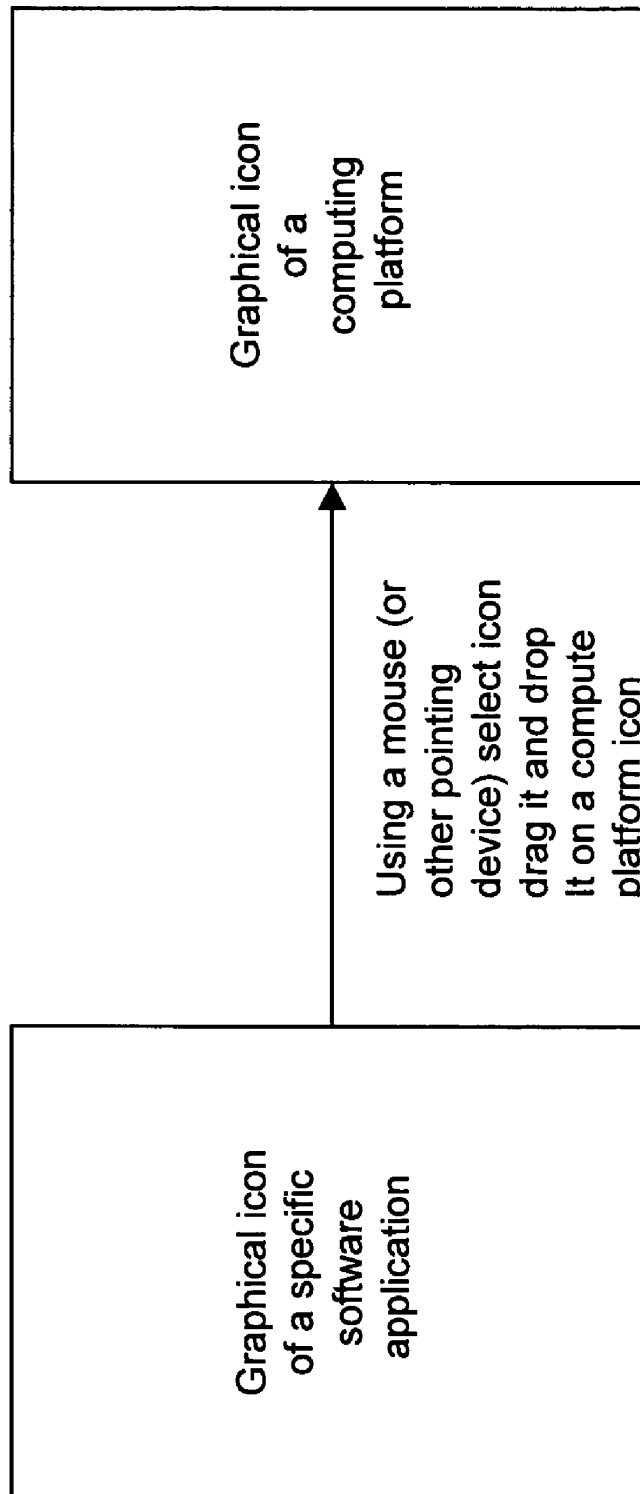


Figure 12

U.S. Patent

Apr. 14, 2009

Sheet 13 of 17

US 7,519,814 B2

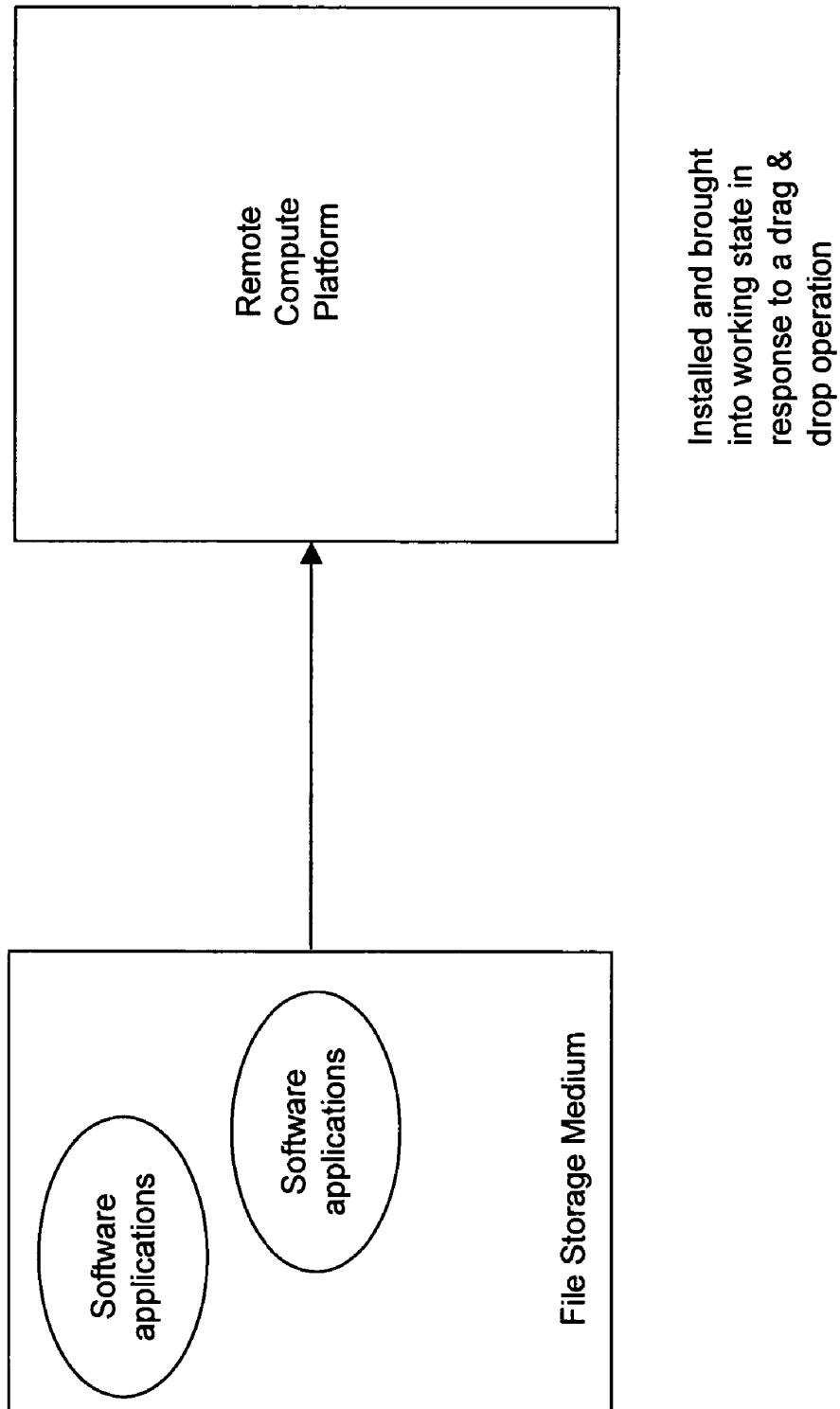


Figure 13

U.S. Patent

Apr. 14, 2009

Sheet 14 of 17

US 7,519,814 B2

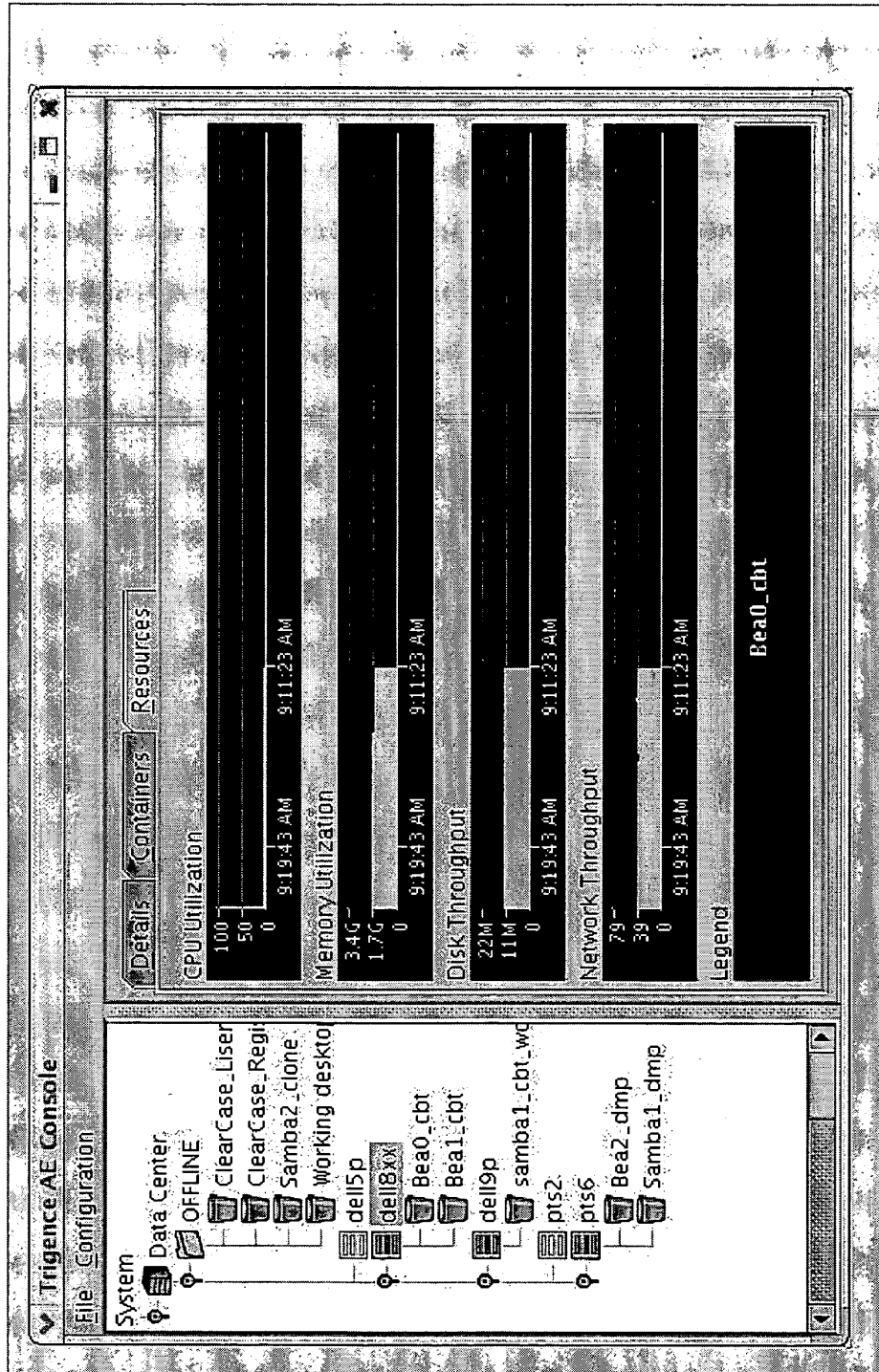


Figure 14

U.S. Patent

Apr. 14, 2009

Sheet 15 of 17

US 7,519,814 B2

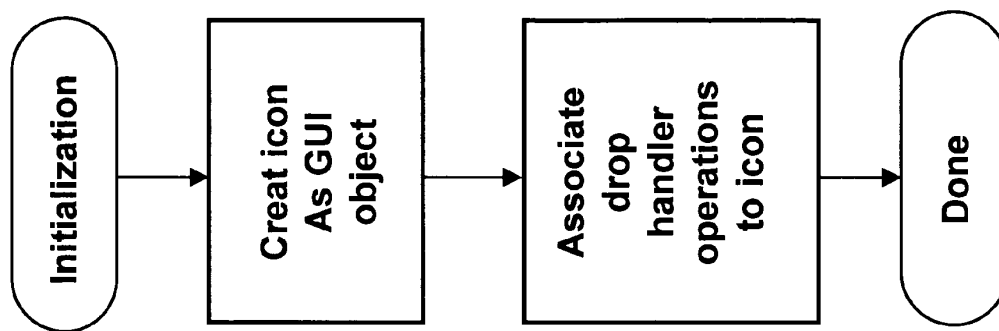


Figure 15

U.S. Patent

Apr. 14, 2009

Sheet 16 of 17

US 7,519,814 B2

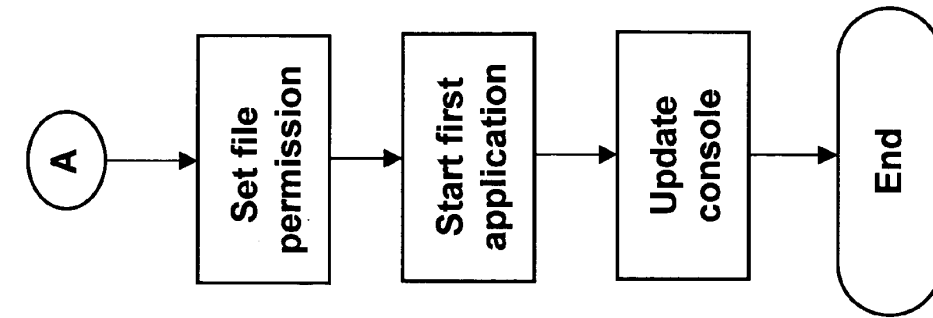
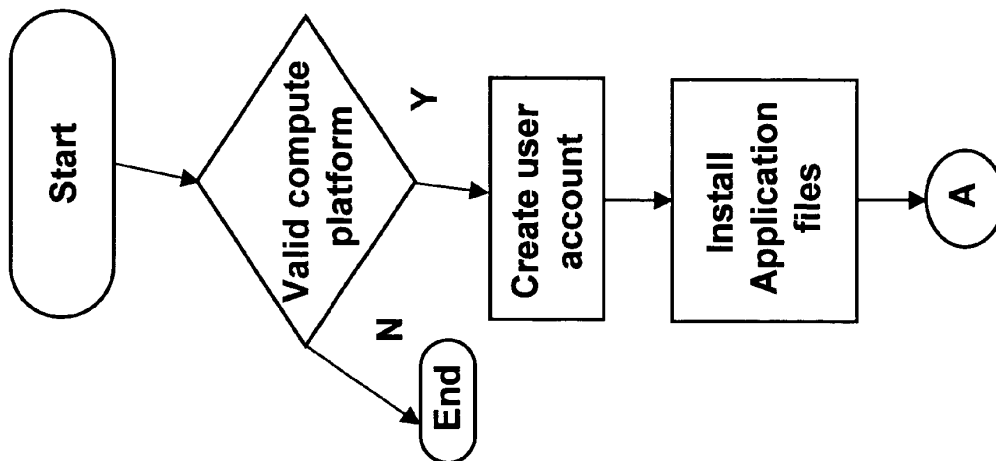


Figure 16



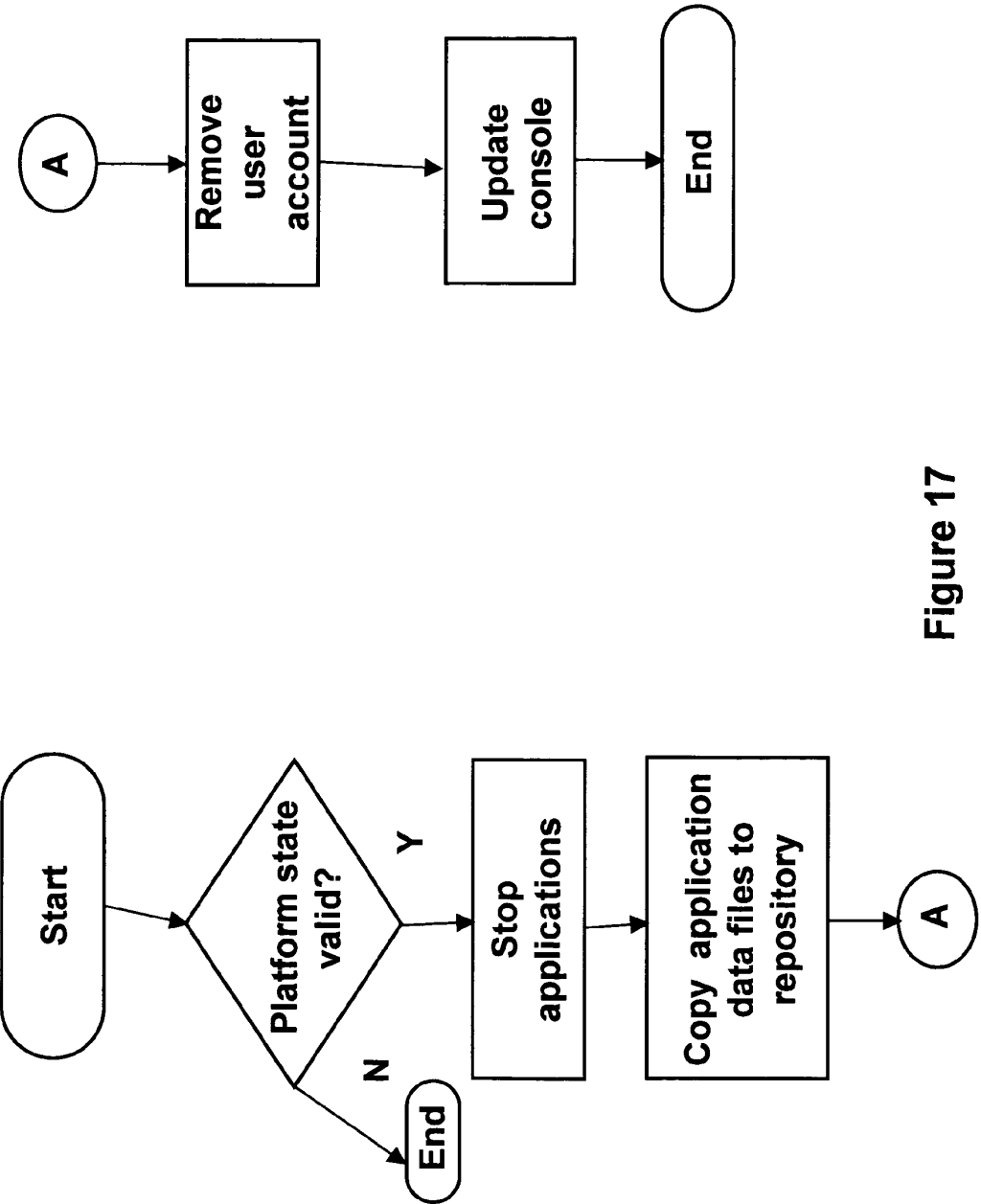


Figure 17

US 7,519,814 B2

1

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**CROSS-REFERENCE TO RELATED APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

2

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

```
/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1
```

US 7,519,814 B2

3

/lib/libnsl.so.1
 /lib/libdl.so.2
 /lib/i686/libpthread.so.0
 /lib/i686/libc.so.6
 /lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
 /etc/httpd/conf
 /etc/httpd/conf.d
 /etc/httpd/logs
 /etc/httpd/modules
 /etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

4

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySQL and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

US 7,519,814 B2

5

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of

6

this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed “spoofing” and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. 3 is a pictorial diagram illustrating the creation of a container.

FIG. 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the “kernel module” which is used to handle system calls.

FIG. 5 is a diagram similar to FIG. 4, wherein additional configuration data is provided.

FIG. 6 is a diagram illustrating a system wherein the containers are secure containers.

FIG. 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. 9 is a diagram that illustrates the installation of a container on a server.

FIG. 10 is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. 14 is a screen snapshot of an implementation according to the schematic of FIG. 13.

FIG. 15 is a flow chart of a method of initializing icons of FIG. 14.

FIG. 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. 14; and,

FIG. 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13.

DETAILED DESCRIPTION

Turning now to FIG. 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c.

US 7,519,814 B2

7

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. 2 illustrates a system in accordance with the invention in which multiple applications **21a**, **21b**, **21c**, **21d**, **21e**, and **21f** can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20a**, **20b** and **20c**. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20a**, **20b** and **20c** are shown in FIG. 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. 4 illustrates a system having two secure containers **20a** **20b**, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20a**, **20b** environment system calls are intercepted, by the kernel module **43** passing through services installed as part of

8

this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

As introduced heretofore, “spoofing” is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system’s kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the ‘uname’ system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has ‘spoofed’ the ‘uname’ system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same “spoofing” mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

- tracking applications, for example a tracking of which applications are in and out of a specific container;
- spoofing particular values returned by the kernel;
- applying resource checks and imposing resource limits;
- monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,
- retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20a** or **20b**. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. 6 that each application executing associated with a container **20a** or **20b**, or contained within a

US 7,519,814 B2

9

container **20a** or **20b**, is able to access files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20a** association are not able to access the files contained in another container **20b**.

When a container **20a** or **20b** is installed specific applications are started, as is shown in FIG. 7 and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. 7, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. 8, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. 9 shows that the files **82a**, **84a** are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing

10

the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. 9.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform and then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. 3, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

US 7,519,814 B2

11

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

- modifying the contents a file to add a container specific IP address;
- modifying the contents of a directory to define start scripts that should be executed;
- modifying the contents of a file to define which mount points will relate to a container;
- removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications

12

are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

US 7,519,814 B2

13

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

14

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

US 7,519,814 B2

15

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. 14, a screen snapshot of an implementation is shown according to the schematic of FIG. 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. 15, shown is a flow chart of a method of initializing the icons of FIG. 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. 16 are performed by an install drop handler. During installation, verification is performed to

16

determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. 15), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

US 7,519,814 B2

17

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

18

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

- a) running an instance of a service on a server;
- b) determining which files are being used; and,
- c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim 14 comprising the steps of:

- assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
- installing the container on a server; and,
- testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

- using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with a unique icon representing

US 7,519,814 B2

19

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim 20 further comprising starting a distributed software application.

22. A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim 17 further comprising creating a user account for the service.

25. A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim 1, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;
displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

20

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim 31, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

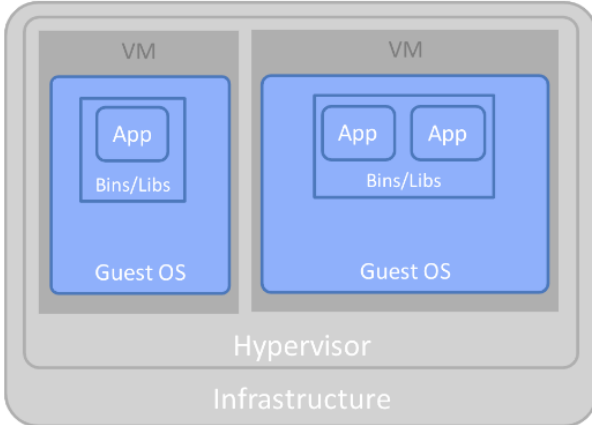
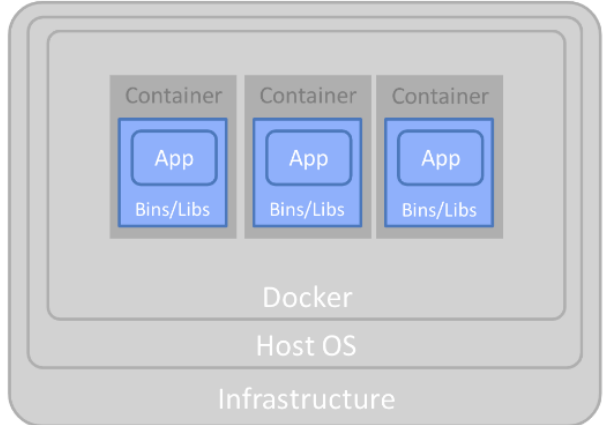
Exhibit 2

U.S. Patent No. 7,519,814 (“814 Patent”)

Accused Instrumentalities: IBM’s Cloud Kubernetes Service (IKS) and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

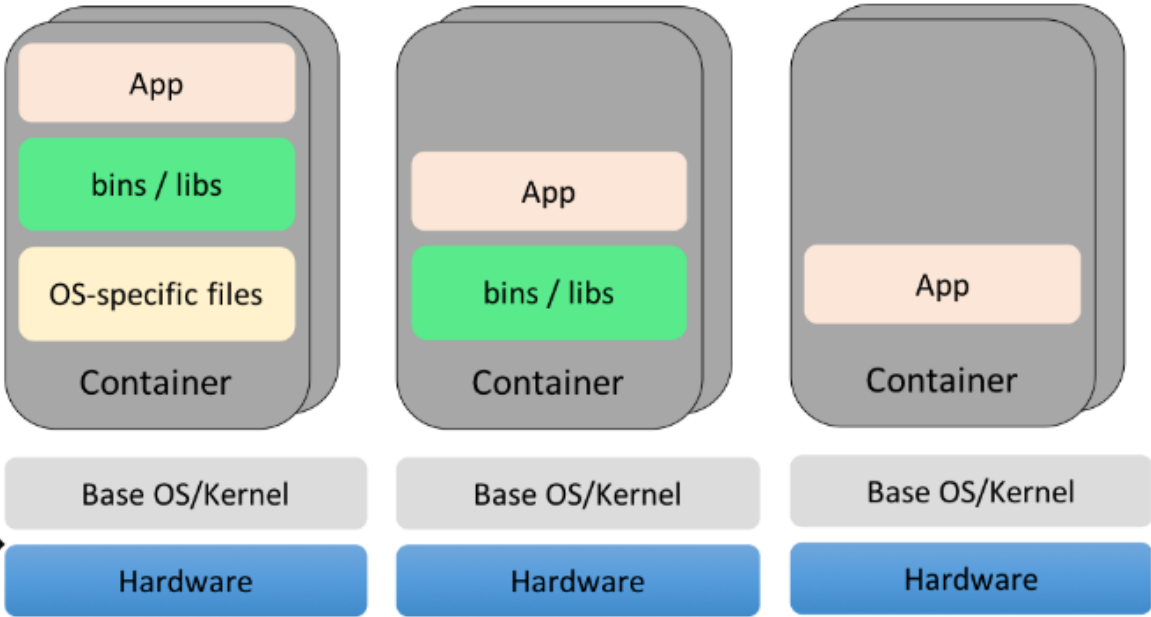
Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>To the extent the preamble is limiting, IBM practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>IBM Cloud® Kubernetes Service provides a fully managed container service for Docker (OCI) containers, so clients can deploy containerized apps onto a pool of compute hosts and subsequently manage those containers. Containers are automatically scheduled and placed onto available compute hosts based on your requirements and availability in the cluster.</p> <p>https://www.ibm.com/products/kubernetes-service</p> <p>With IBM Cloud Kubernetes Service, you can deploy Docker containers into pods that run on your worker nodes. The worker nodes come with a set of add-on pods to help you manage your containers. Install more add-ons through Helm, a Kubernetes package manager. These add-ons can extend your apps with dashboards, logging, IBM Cloud and IBM Watson® services and more.</p> <p>https://www.ibm.com/products/kubernetes-service</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="661 332 1680 487">Docker is an open source platform that enables developers to build, deploy, run, update and manage <i>containers</i>—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.</p> <p data-bbox="651 511 1092 544">https://www.ibm.com/topics/docker</p> <div data-bbox="661 592 1869 1079"><div><p data-bbox="840 597 1081 630">Virtual Machines</p><p>The diagram shows two Virtual Machines (VMs) stacked vertically. Each VM contains a Guest OS, which in turn contains Bins/Libs and an App. The VMs are managed by a Hypervisor, which sits on top of the Infrastructure layer.</p></div><div><p data-bbox="1491 597 1648 630">Containers</p><p>The diagram shows three Containers stacked vertically. Each Container contains Bins/Libs and an App. The Containers are managed by Docker, which sits on top of the Host OS, which in turn sits on top of the Infrastructure layer.</p></div></div> <p data-bbox="651 1112 1564 1144">https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/</p>

Claim 1	Accused Instrumentalities
	<p>Containers are executable units of software in which application code is packaged along with its libraries and dependencies, in common ways so that the code can be run anywhere—whether it be on desktop, traditional IT or the cloud.</p> <p>To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (e.g. Linux namespaces and cgroups, Windows silos and job objects) can be leveraged to isolate processes and control the amount of CPU, memory and disk that those processes can access.</p> <p>Containers are small, fast and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can instead simply leverage the features and resources of the host OS.</p> <p>https://www.ibm.com/topics/containers</p> <p>With containers, you can isolate the ecosystem to run an application on any host OS (operating system). Containers can wrap code, runtimes, system tools, system libraries—everything that can be installed on a server. Containers are like virtual machines (VMs), but with a key difference in their architectural approach. Images that run on VMs have a full copy of the guest OS, including the necessary binaries and libraries. Images that run on containers share the OS kernel on the host.</p> <p>The Docker Engine builds and spins images on the containers. The engine is a lightweight container runtime that can run on almost any OS. You can run a container anywhere that a Docker Engine can be installed—on bare metal servers, clouds, and even inside a VM. You can move containers from one environment to another without recoding the application.</p> <p>Containers can help DevOps teams in three ways:</p> <ul style="list-style-type: none"> • Increase development productivity by reducing the time spent on environment setup • Eliminate issues that are caused by software dependencies • Avoid inconsistencies when applications are run in different environments <p>You can use IBM Cloud Kubernetes Service to run containers on IBM Cloud.</p> <p>https://www.ibm.com/garage/method/practices/run/tool_ibm_container/, last accessed on Nov. 17, 2023.</p>

Claim 1	Accused Instrumentalities
	<p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p> <p>Today Docker containerization also works with Microsoft Windows and Apple MacOS. Developers can run Docker containers on any operating system, and most leading cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to help developers build, deploy and run applications containerized with Docker.</p> <p>https://www.ibm.com/topics/docker</p>

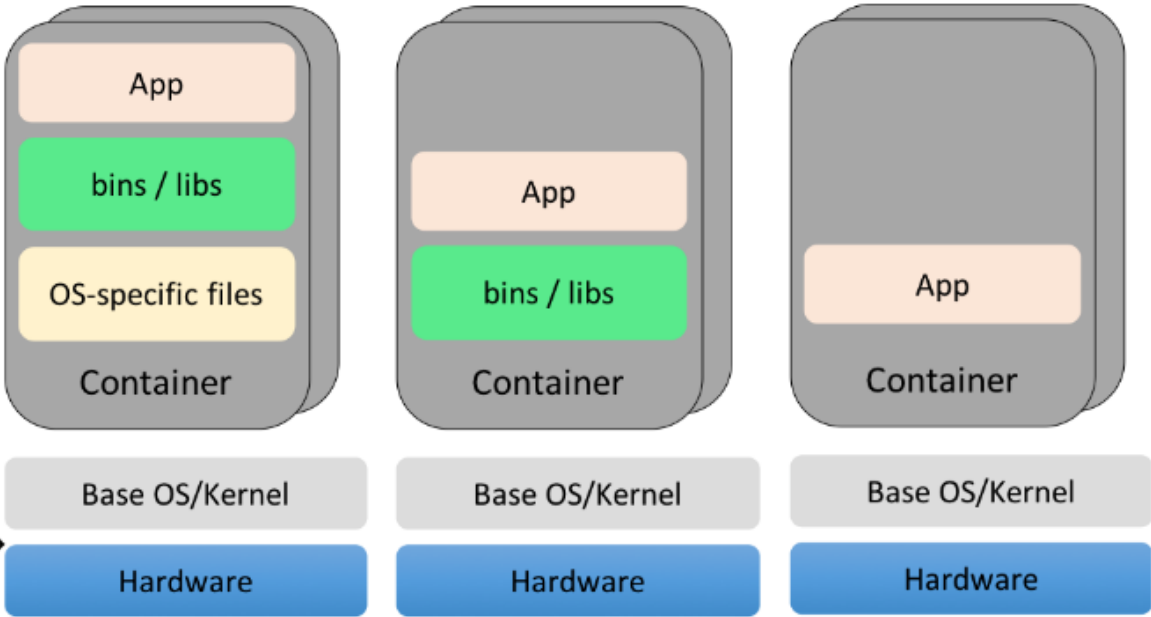
Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<div style="text-align: center;"> <p><u>Container</u> <u>Container</u> <u>Container</u></p>  <p>Containers share the same base Kernel</p> <p>https://ibm.github.io/kube101/</p> </div>
[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application	The method practiced by IBM through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of

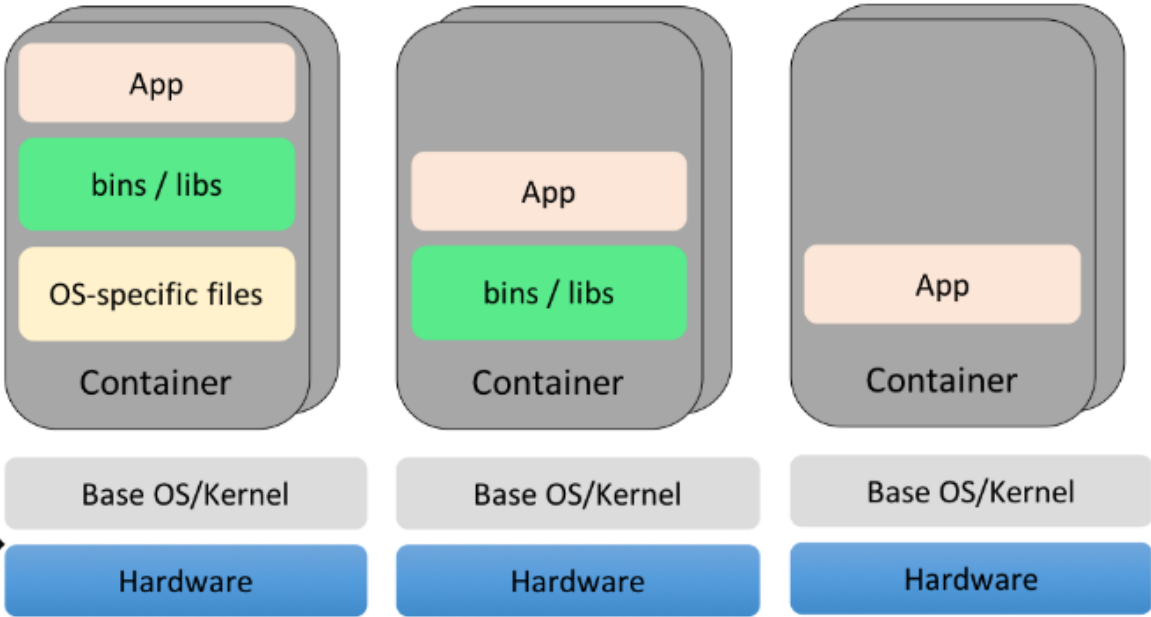
Claim 1	Accused Instrumentalities
<p>software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p> <p><i>See, e.g.:</i></p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p> <p>Today Docker containerization also works with Microsoft Windows and Apple MacOS. Developers can run Docker containers on any operating system, and most leading cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to help developers build, deploy and run applications containerized with Docker.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="659 331 1801 1187"> <p>The diagram illustrates three containers, each labeled 'Container', stacked on a shared 'Base OS/Kernel' layer, which sits on 'Hardware'. Each container contains an 'App' and 'bins / libs'. The first container also includes 'OS-specific files'. Below the containers, text states 'Containers share the same base Kernel'.</p> </div> <p>https://ibm.github.io/kube101/</p>
[1b] wherein the set of associated system files are compatible with a local kernel	In the method practiced by IBM through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.

Claim 1	Accused Instrumentalities
of at least some of the plurality of different operating systems,	<p data-bbox="653 319 766 354"><i>See, e.g.:</i></p> <p data-bbox="653 386 1833 634">Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p data-bbox="653 670 1839 967">Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.</p> <p data-bbox="653 987 1201 1021">https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<div style="text-align: center;"> <p><u>Container</u> <u>Container</u> <u>Container</u></p>  <p>Containers share the same base Kernel</p> <p>https://ibm.github.io/kube101/</p> </div>
[1c] the containers of application software excluding a kernel,	<p>In the method practiced by IBM through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<div style="text-align: center;"> <p><u>Container</u> <u>Container</u> <u>Container</u></p>  <p>Containers share the same base Kernel</p> <p>https://ibm.github.io/kube101/</p> </div>
[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the	In the method practiced by IBM through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.

Claim 1	Accused Instrumentalities
associated local system files that remain resident on the server,	<p><i>See, e.g.:</i></p> <p>Rather than spinning up an entire virtual machine, containerization packages together everything needed to run a single application or microservice (along with runtime libraries they need to run). The container includes all the code, its dependencies and even the operating system itself. This enables applications to run almost anywhere — a desktop computer, a traditional IT infrastructure or the cloud.</p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p>
[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,	<p>In the method practiced by IBM through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.</p> <p>Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<p>With containers, you can isolate the ecosystem to run an application on any host OS (operating system). Containers can wrap code, runtimes, system tools, system libraries—everything that can be installed on a server. Containers are like virtual machines (VMs), but with a key difference in their architectural approach. Images that run on VMs have a full copy of the guest OS, including the necessary binaries and libraries. Images that run on containers share the OS kernel on the host.</p> <p>The Docker Engine builds and spins images on the containers. The engine is a lightweight container runtime that can run on almost any OS. You can run a container anywhere that a Docker Engine can be installed—on bare metal servers, clouds, and even inside a VM. You can move containers from one environment to another without recoding the application.</p> <p>Containers can help DevOps teams in three ways:</p> <ul style="list-style-type: none"> • Increase development productivity by reducing the time spent on environment setup • Eliminate issues that are caused by software dependencies • Avoid inconsistencies when applications are run in different environments <p>You can use IBM Cloud Kubernetes Service to run containers on IBM Cloud.</p> <p>https://www.ibm.com/garage/method/practices/run/tool_ibm_container/, last accessed on Nov. 17, 2023.</p>
[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,	<p>In the method practiced by IBM through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p><i>See, e.g.:</i></p> <p>Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple virtual machines (VMs) to share the CPU, memory and other resources of a single hardware server.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p>Fault isolation: Each containerized application is isolated and operates independently of others. The failure of one container does not affect the continued operation of any other containers. Development teams can identify and correct any technical issues within one container without any downtime in other containers. Also, the container engine can leverage any OS security isolation techniques—such as SELinux access control—to isolate faults within containers.</p> <p>https://www.ibm.com/topics/containerization</p>
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p><i>See, e.g.:</i></p> <p>Limit the number of privileged containers. Containers run as a separate Linux process on the compute host that is isolated from other processes. Although users have root access inside the container, the permissions of this user are limited outside the container to protect other Linux processes, the host file system, and host devices. Some apps require access to the host file system or advanced permissions to run properly. You can run containers in privileged mode to allow the container the same access as the processes running on the compute host. Keep in mind that privileged containers can cause huge damage to the cluster and the underlying compute host if they become compromised. Try to limit the number of containers that run in privileged mode and consider changing the configuration for your app so that the app can run without advanced permissions.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-security</p>

Claim 1	Accused Instrumentalities
	<p>Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple virtual machines (VMs) to share the CPU, memory and other resources of a single hardware server.</p> <p>https://www.ibm.com/topics/docker</p>

Exhibit 3



(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,784,058 B2**

(45) **Date of Patent:** **Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**

2002/0004854 A1 1/2002 Hartley
2002/0174215 A1 11/2002 Schaefer 709/224
2003/0101292 A1 5/2003 Fisher
2004/0025165 A1* 2/2004 Desoli et al. 719/310

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002
WO WO 2006/039181 A 4/2006

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

OTHER PUBLICATIONS

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1293 days.

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

* cited by examiner

(21) Appl. No.: **10/946,536**

Primary Examiner—Hyung S Sough

Assistant Examiner—Syed Roni

(22) Filed: **Sep. 21, 2004**

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A. Attorneys at Law

(65) **Prior Publication Data**

US 2005/0066303 A1 Mar. 24, 2005

(57) **ABSTRACT**

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep. 22, 2003.

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

(51) **Int. Cl.**
G06F 9/22 (2006.01)

(52) **U.S. Cl.** **719/310**; 719/319

(58) **Field of Classification Search** 719/310,
719/319

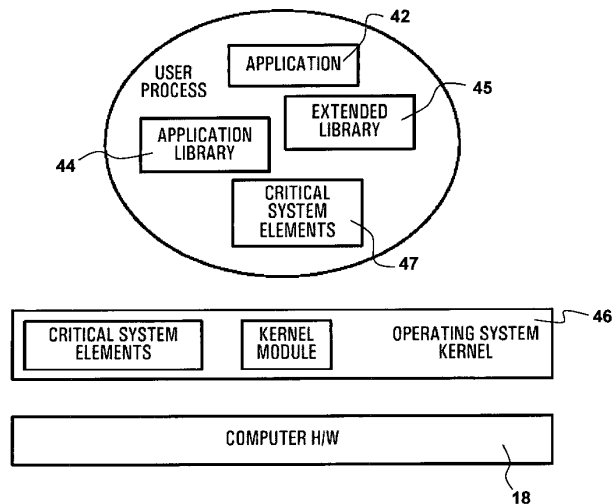
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,481,706 A * 1/1996 Peek 710/200
6,212,574 B1 * 4/2001 O'Rourke et al. 719/321
6,260,075 B1 * 7/2001 Cabrero et al. 719/310

18 Claims, 7 Drawing Sheets



U.S. Patent

Aug. 24, 2010

Sheet 1 of 7

US 7,784,058 B2

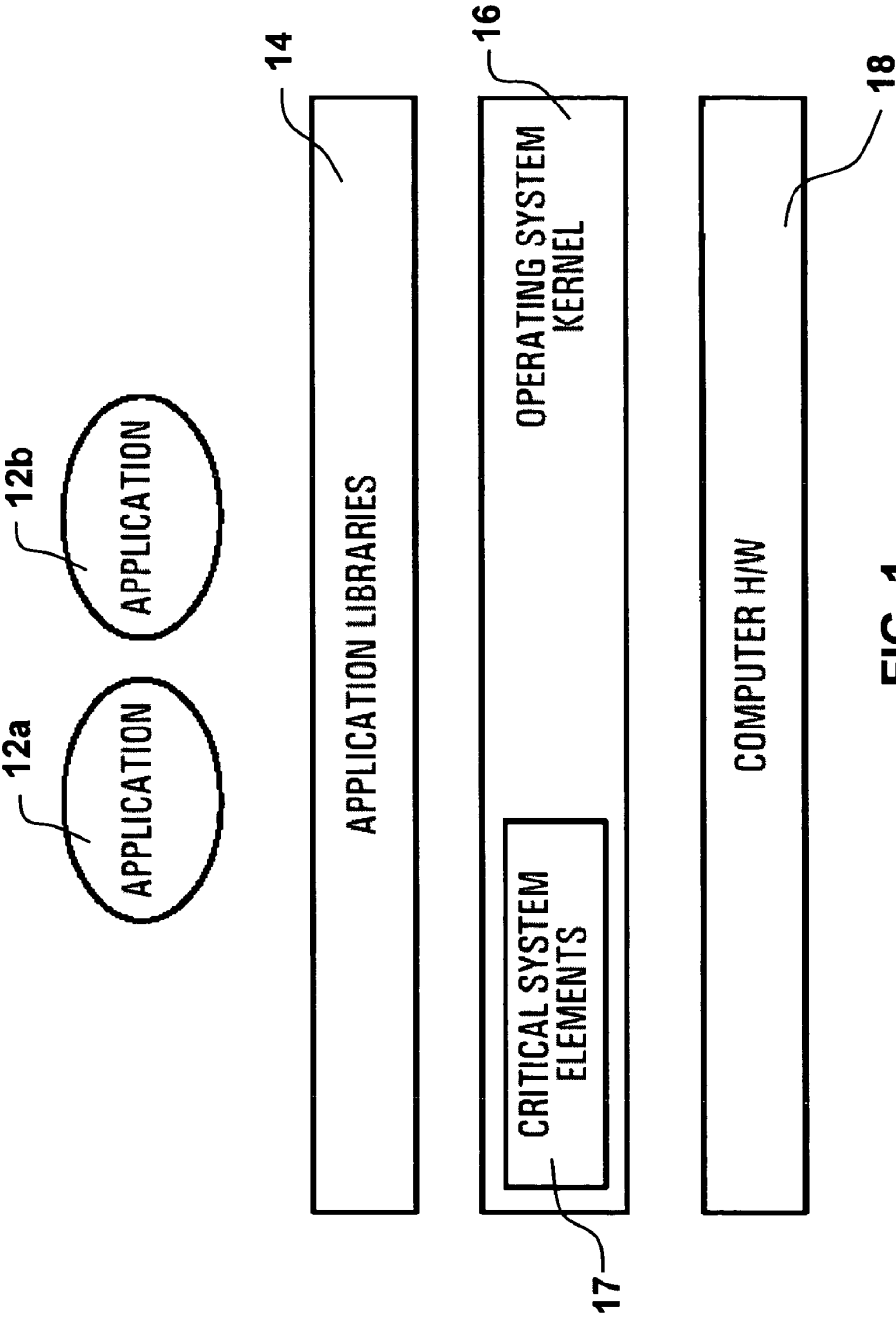


FIG. 1
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 2 of 7

US 7,784,058 B2

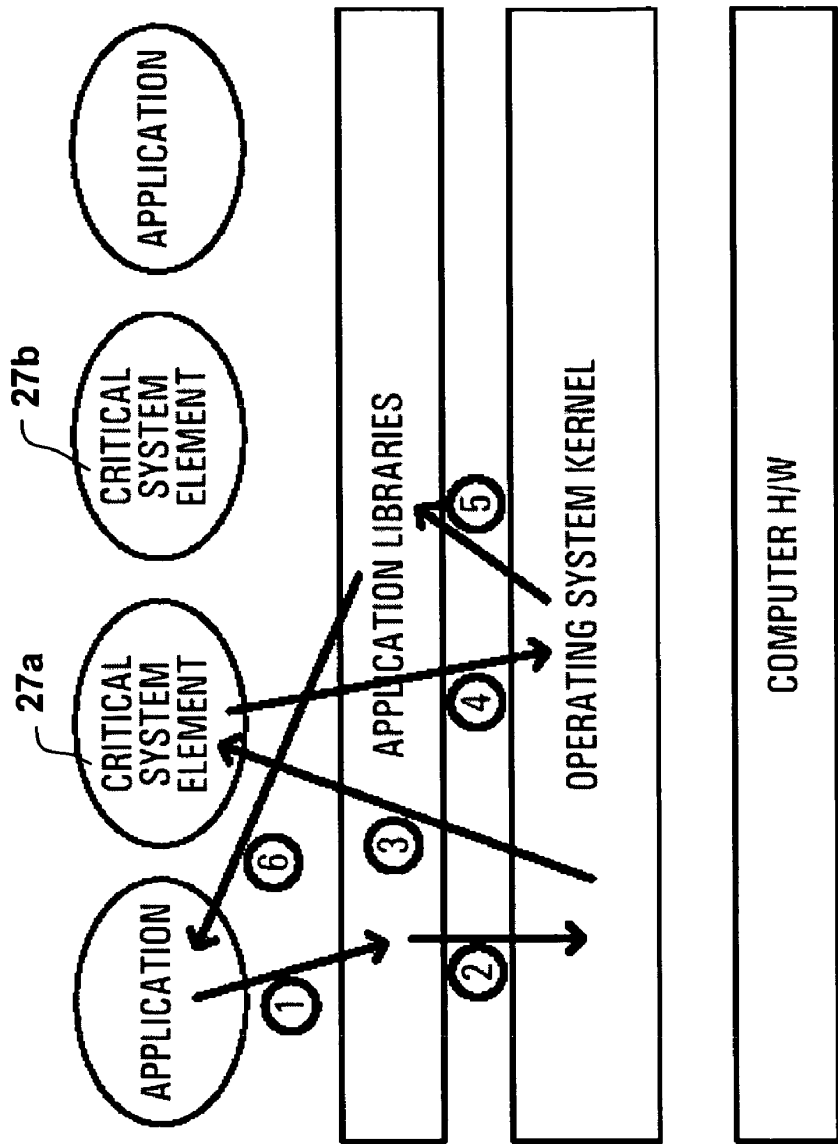


FIG. 2a
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 3 of 7

US 7,784,058 B2

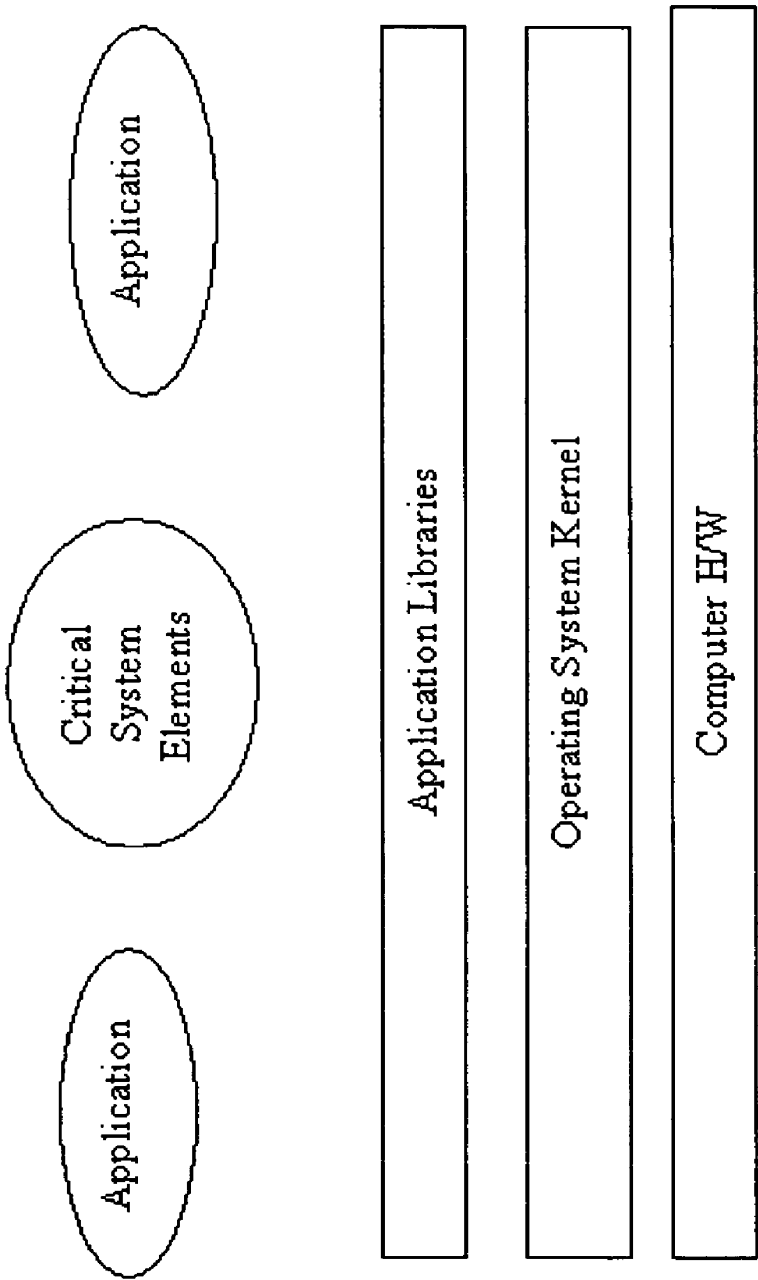


FIG. 2b
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 4 of 7

US 7,784,058 B2

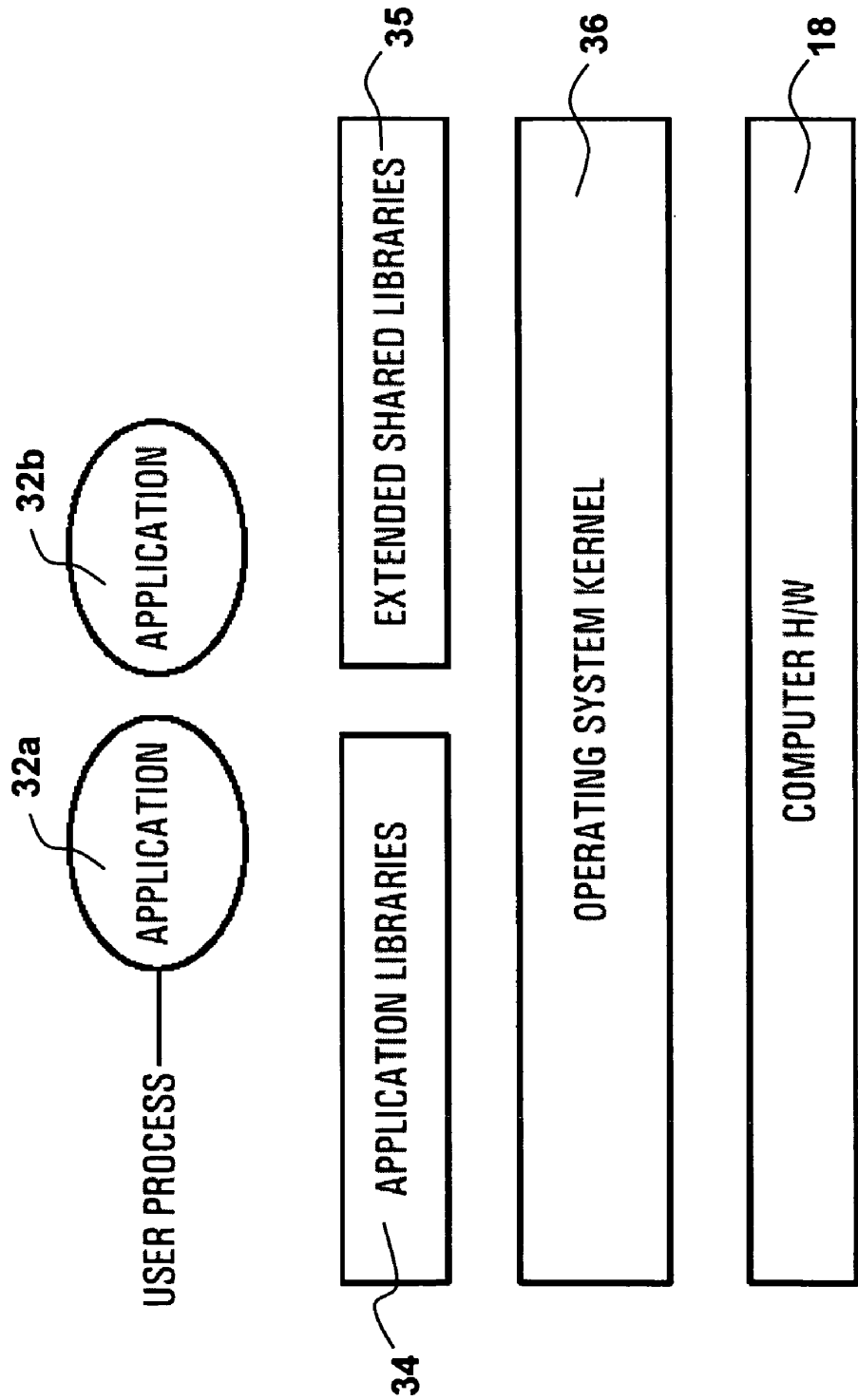


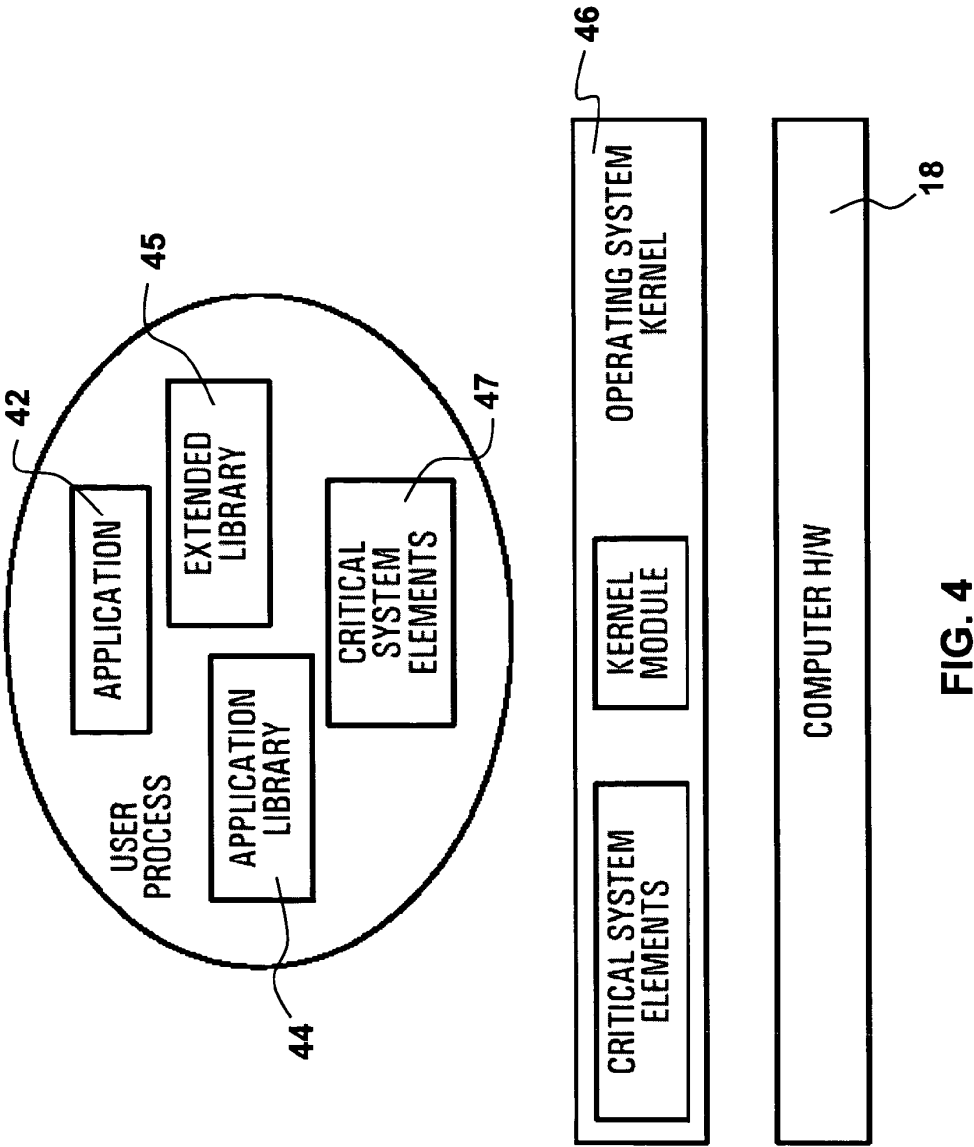
FIG. 3

U.S. Patent

Aug. 24, 2010

Sheet 5 of 7

US 7,784,058 B2



U.S. Patent

Aug. 24, 2010

Sheet 6 of 7

US 7,784,058 B2

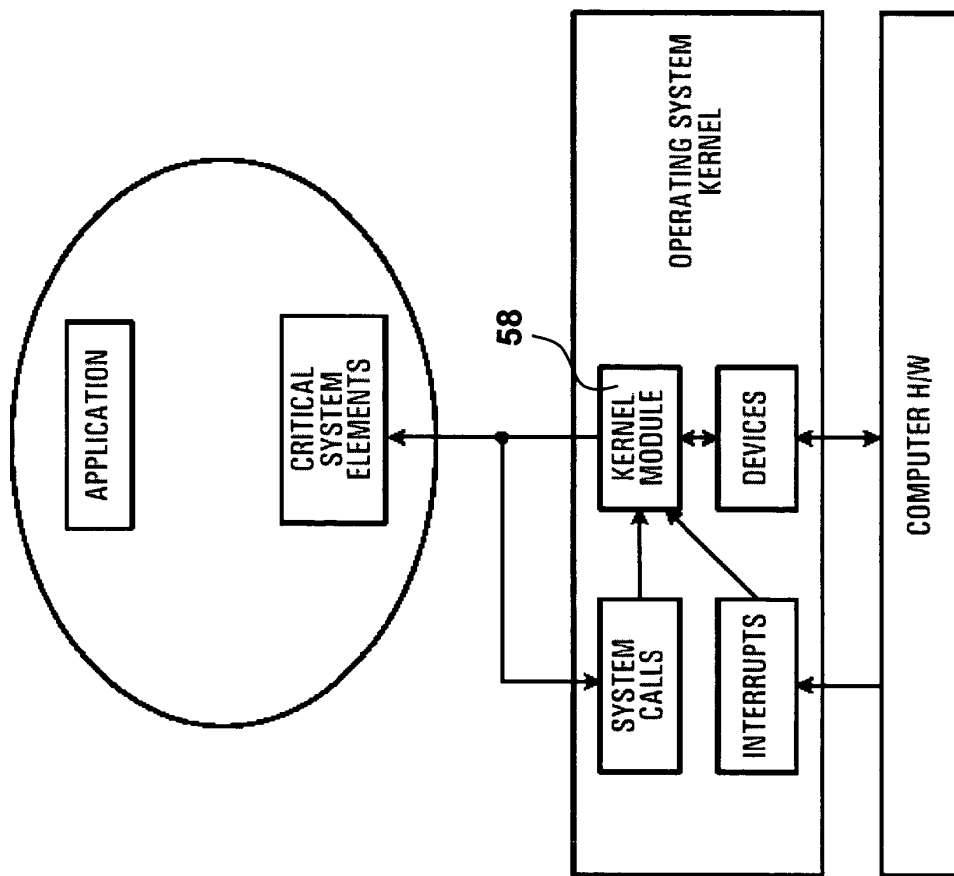


FIG. 5

U.S. Patent

Aug. 24, 2010

Sheet 7 of 7

US 7,784,058 B2

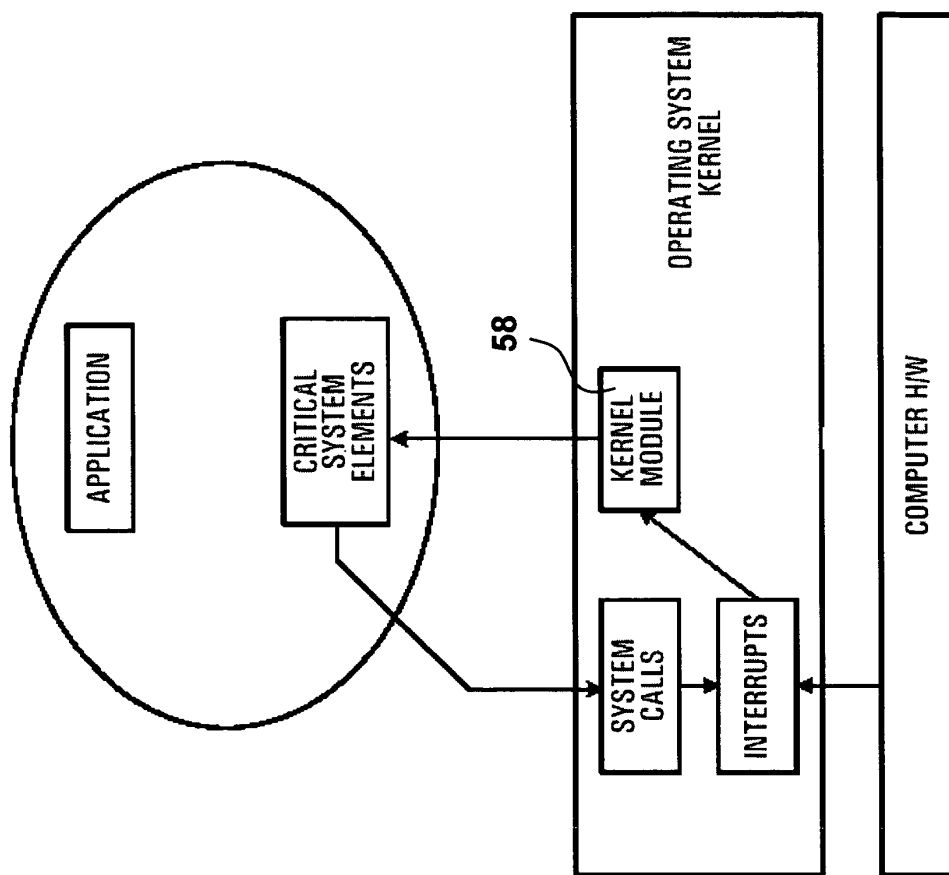


FIG. 6

US 7,784,058 B2

1

**COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled “User Mode Critical System Element as Shared Libs”, which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

2

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.

2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

US 7,784,058 B2

3

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

4

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. 1 is an architectural view of the traditional monolithic prior art operating system;

US 7,784,058 B2

5

FIG. 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. 3 is an architectural view of an embodiment of the invention;

FIG. 4 is a functional view showing how critical system elements exist in the same context as an application;

FIG. 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. 6 shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TCP/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP

6

stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality

2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided

2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b

US 7,784,058 B2

7

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12a**, **12b** makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12a**, **12b** in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2a** shows a system architecture where critical system elements **27a**, **27b** execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2a** is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2a** as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2b** is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2a** and FIG. **2b** is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2b** is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the

8

case with a multi-server architecture depicted in FIG. **2a**. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32a**, **32b**, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45,47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

US 7,784,058 B2

9

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. 4. FIG. 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

10

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:

- a) a processor;
- b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,
- c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

US 7,784,058 B2

11

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

6. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

7. A computing system according to claim 6 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

8. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

12

9. A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

10. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

11. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

12. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

13. A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

14. A computing system according to claim 13 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

15. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

16. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

17. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.

* * * * *

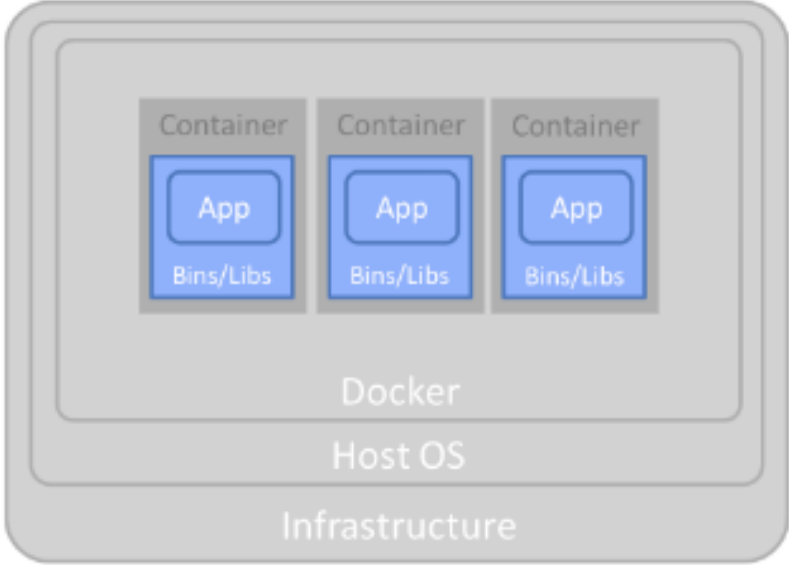
Exhibit 4

U.S. Patent No. 7,784,058 (“’058 Patent”)

Accused Instrumentalities: IBM’s IBM Cloud Kubernetes Service, and all versions and variations thereof since the issuance of the asserted patent.

Claim 1






Claim 1	Accused Instrumentalities
<p>[1pre] 1. A computing system for executing a plurality of software applications comprising:</p>	<p>To the extent the preamble is limiting, each Accused Instrumentality comprises or constitutes a computing system for executing a plurality of software applications as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>IBM Cloud® Kubernetes Service provides a fully managed container service for Docker (OCI) containers, so clients can deploy containerized apps onto a pool of compute hosts and subsequently manage those containers. Containers are automatically scheduled and placed onto available compute hosts based on your requirements and availability in the cluster.</p> <p>https://www.ibm.com/products/kubernetes-service</p> <p>With IBM Cloud Kubernetes Service, you can deploy Docker containers into pods that run on your worker nodes. The worker nodes come with a set of add-on pods to help you manage your containers. Install more add-ons through Helm, a Kubernetes package manager. These add-ons can extend your apps with dashboards, logging, IBM Cloud and IBM Watson® services and more.</p> <p>https://www.ibm.com/products/kubernetes-service</p>

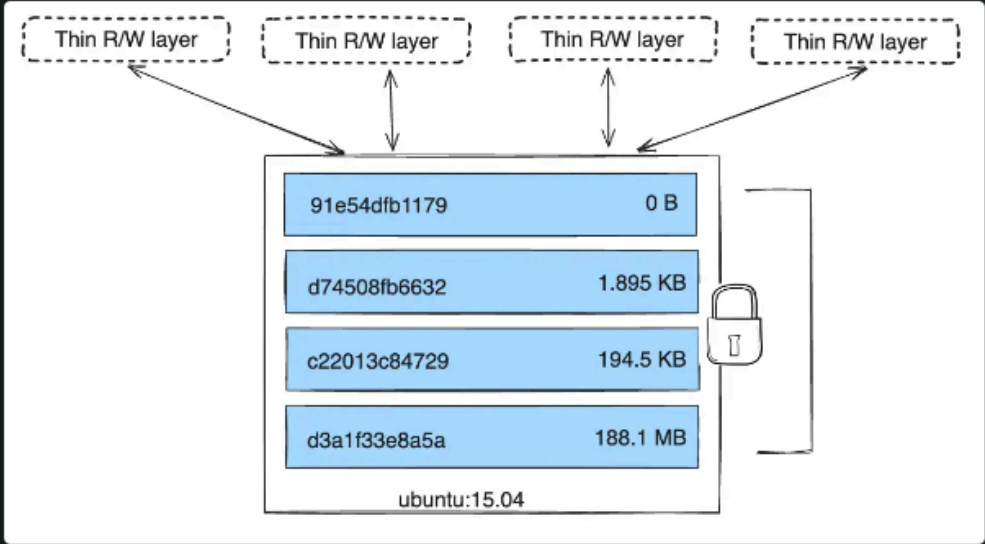
Claim 1	Accused Instrumentalities
	<p data-bbox="961 358 1163 402">Containers</p>  <p>The diagram illustrates a container architecture stack. At the base is a grey rounded rectangle labeled 'Infrastructure'. Above it is a grey rounded rectangle labeled 'Host OS'. Above the Host OS is a grey rounded rectangle labeled 'Docker'. Inside the Docker container are three separate grey rounded rectangles, each labeled 'Container'. Each 'Container' contains a blue rounded rectangle labeled 'App' and a smaller blue rounded rectangle labeled 'Bins/Libs' below it.</p> <p data-bbox="653 1073 1568 1105">https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/</p>
[1a] a) a processor;	<p data-bbox="653 1141 1299 1174">Each Accused Instrumentality comprises a processor.</p> <p data-bbox="653 1203 764 1235"><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containers are executable units of software in which application code is packaged along with its libraries and dependencies, in common ways so that the code can be run anywhere—whether it be on desktop, traditional IT or the cloud.</p> <p>To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (e.g. Linux namespaces and cgroups, Windows silos and job objects) can be leveraged to isolate processes and control the amount of CPU, memory and disk that those processes can access.</p> <p>Containers are small, fast and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can instead simply leverage the features and resources of the host OS.</p> <p>https://www.ibm.com/topics/containers</p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p>

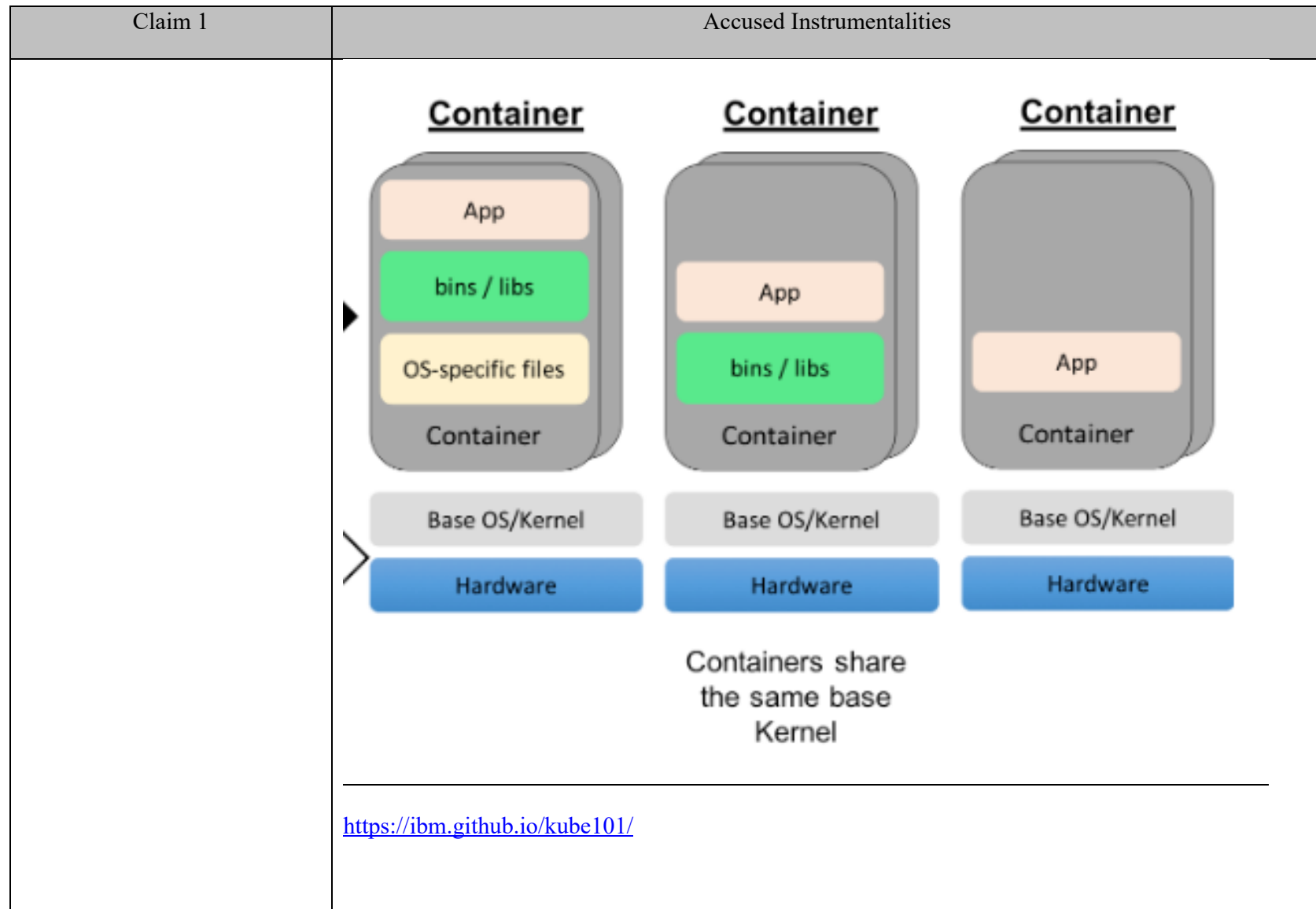
Claim 1	Accused Instrumentalities
<p>[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,</p>	<p>Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.</p> <p><i>See, e.g.:</i></p> <p>Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.</p> <p>Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.</p> <p>https://www.ibm.com/topics/containerization</p> <p>Kernel mode</p> <p>Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p>https://www.techtarget.com/searchdatacenter/definition/kernel</p>

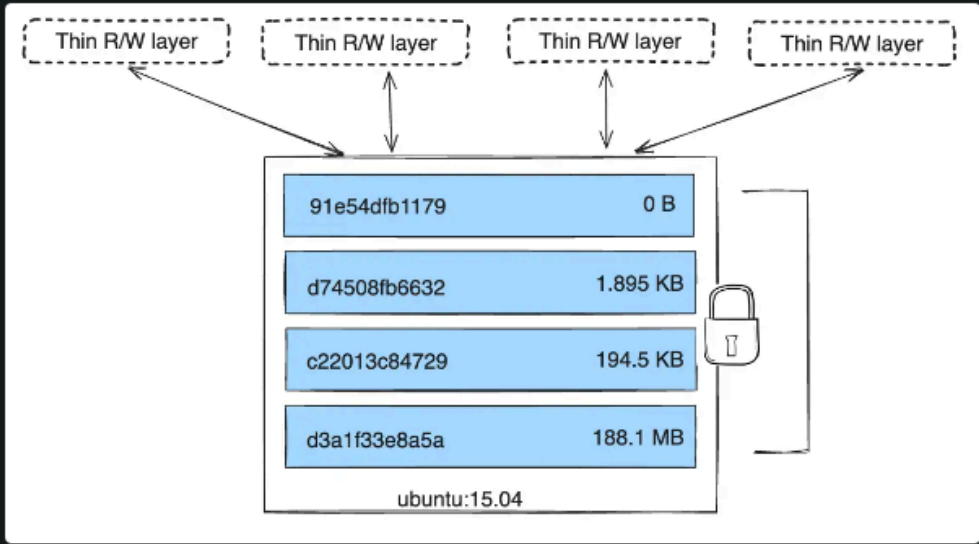
Claim 1	Accused Instrumentalities
<p>[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and</p>	<p>Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.</p> <p><i>See, e.g.:</i></p> <p>A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p>An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-images</p>

Claim 1	Accused Instrumentalities		
	Docker base image	Supported versions	Source of security notices
	Alpine	All stable versions with vendor security support.	Alpine SecDB database  .
	Debian	All stable versions with vendor security support. CVEs on binary packages that are associated with the Debian source package <code>linux</code> , such as <code>linux-libc-dev</code> , are not reported. Most of these binary packages are kernel and kernel modules, which are not run in container images.	Debian Security Bug Tracker  .
	GoogleContainerTools distroless	All stable versions with vendor security support.	GoogleContainerTools distroless  .
	Red Hat® Enterprise Linux® (RHEL)	RHEL/UBI 7, RHEL/UBI 8, and RHEL/UBI 9	Red Hat Security Data API  .
	Ubuntu	All stable versions with vendor security support.	Ubuntu CVE Tracker  .
	https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&interface=ui		

Claim 1	Accused Instrumentalities
	<p data-bbox="667 342 1869 464">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p data-bbox="653 1127 1213 1157">https://docs.docker.com/storage/storagedriver/</p>





Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>Docker images are made up of layers, and each layer corresponds to a version of the image. Whenever a developer makes changes to the image, a new top layer is created, and this top layer replaces the previous top layer as the current version of the image. Previous layers are saved for rollbacks or to be re-used in other projects.</p> <p>Each time a container is created from a Docker image, yet another new layer called the container layer is created. Changes made to the container—such as the addition or deletion of files—are saved to the container layer only and exist only while the container is running. This iterative image-creation process enables increased overall efficiency since multiple live container instances can run from just a single base image, and when they do so, they leverage a common stack.</p> <p>https://www.ibm.com/topics/docker</p>



Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>
[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of	In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.

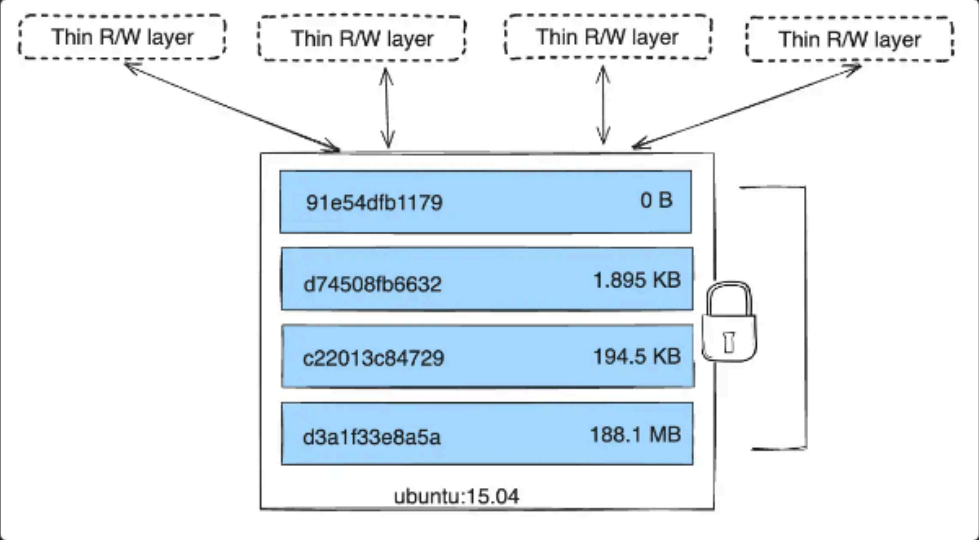
Claim 1	Accused Instrumentalities
<p>software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,</p>	<p>For example, a Docker base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and dependencies, including system libraries or shared library critical system elements (SLCSEs). When the container runs the image, it creates a runtime instance of that container image.</p> <p><i>See, e.g.:</i></p> <p>A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p>An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-images</p>

Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>Docker images are made up of layers, and each layer corresponds to a version of the image. Whenever a developer makes changes to the image, a new top layer is created, and this top layer replaces the previous top layer as the current version of the image. Previous layers are saved for rollbacks or to be re-used in other projects.</p> <p>Each time a container is created from a Docker image, yet another new layer called the container layer is created. Changes made to the container—such as the addition or deletion of files—are saved to the container layer only and exist only while the container is running. This iterative image-creation process enables increased overall efficiency since multiple live container instances can run from just a single base image, and when they do so, they leverage a common stack.</p> <p>https://www.ibm.com/topics/docker</p> <p>Following software is installed on the Docker containers as part of the Product Master image deployment:</p> <ul style="list-style-type: none">– Red Hat Enterprise Linux (RHEL) 7 Universal Base Image (UBI) base Docker image <p>https://www.ibm.com/docs/en/product-master/12.0.0?topic=deployment-installing-product-by-using-docker-images</p>

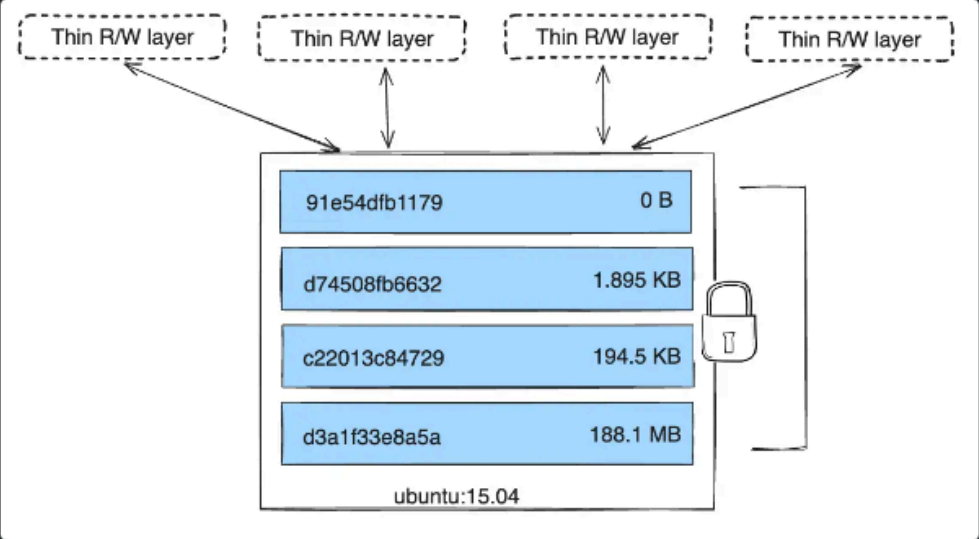
Claim 1	Accused Instrumentalities
	<div><div>ubuntu </div><div>Updated 15 days ago</div><div>Ubuntu is a Debian-based Linux operating system based on free software.</div><div>Linux IBM Z 386 riscv64 x86-64 ARM ARM 64 PowerPC 64 LE</div></div> <div><div>debian </div><div>Updated 35 minutes ago</div><div>Debian is a Linux distribution that's composed entirely of free and open-source software.</div><div>Linux riscv64 x86-64 ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z</div></div> <div>https://hub.docker.com/search?image_filter=official&type=image&q=</div>

Claim 1	Accused Instrumentalities																																																						
	<table><tr><th>Platform</th><th>x86_64 / amd64</th><th>arm64 / aarch64</th><th>arm (32-bit)</th><th>ppc64le</th><th>s390x</th></tr><tr><td>CentOS</td><td>✓</td><td>✓</td><td></td><td>✓</td><td></td></tr><tr><td>Debian</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td></td></tr><tr><td>Fedora</td><td>✓</td><td>✓</td><td></td><td>✓</td><td></td></tr><tr><td>Raspberry Pi OS (32-bit)</td><td></td><td></td><td>✓</td><td></td><td></td></tr><tr><td>RHEL (s390x)</td><td></td><td></td><td></td><td></td><td>✓</td></tr><tr><td>SLES</td><td></td><td></td><td></td><td></td><td>✓</td></tr><tr><td>Ubuntu</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr><tr><td>Binaries</td><td>✓</td><td>✓</td><td>✓</td><td></td><td></td></tr></table> <p>https://docs.docker.com/engine/install/</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p>	Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x	CentOS	✓	✓		✓		Debian	✓	✓	✓	✓		Fedora	✓	✓		✓		Raspberry Pi OS (32-bit)			✓			RHEL (s390x)					✓	SLES					✓	Ubuntu	✓	✓	✓	✓	✓	Binaries	✓	✓	✓		
Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x																																																		
CentOS	✓	✓		✓																																																			
Debian	✓	✓	✓	✓																																																			
Fedora	✓	✓		✓																																																			
Raspberry Pi OS (32-bit)			✓																																																				
RHEL (s390x)					✓																																																		
SLES					✓																																																		
Ubuntu	✓	✓	✓	✓	✓																																																		
Binaries	✓	✓	✓																																																				

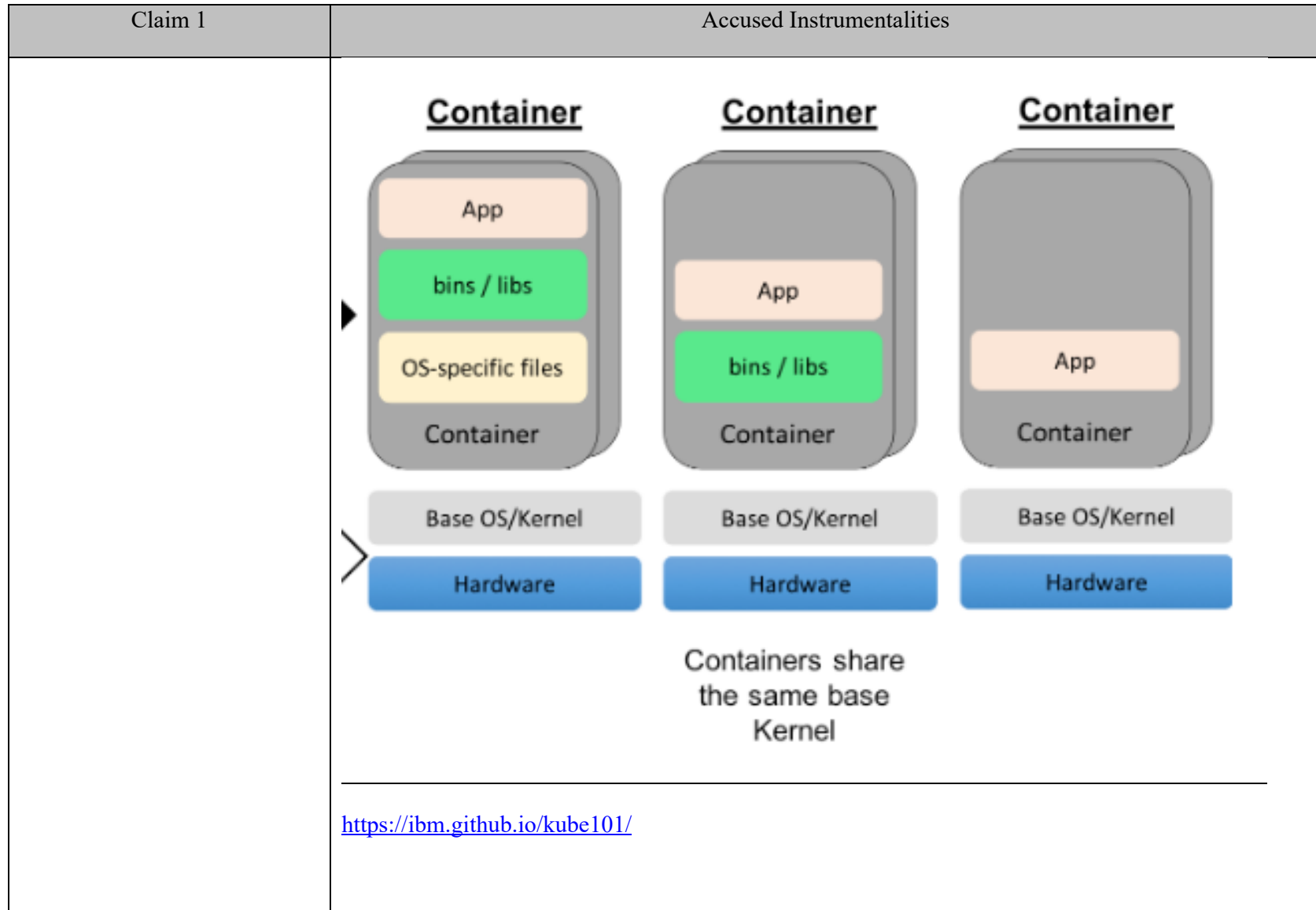
Claim 1	Accused Instrumentalities
	<p data-bbox="898 354 1556 386">A container is a runtime instance of a docker image.</p> <p data-bbox="898 440 1289 472">A Docker container consists of</p> <div data-bbox="678 574 800 607">container</div> <ul data-bbox="911 526 1331 688" style="list-style-type: none"><li data-bbox="911 526 1157 558">• A Docker image<li data-bbox="911 591 1289 623">• An execution environment<li data-bbox="911 656 1331 688">• A standard set of instructions <p data-bbox="655 748 1150 781">https://docs.docker.com/glossary/#image</p>

Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>
[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software	In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.

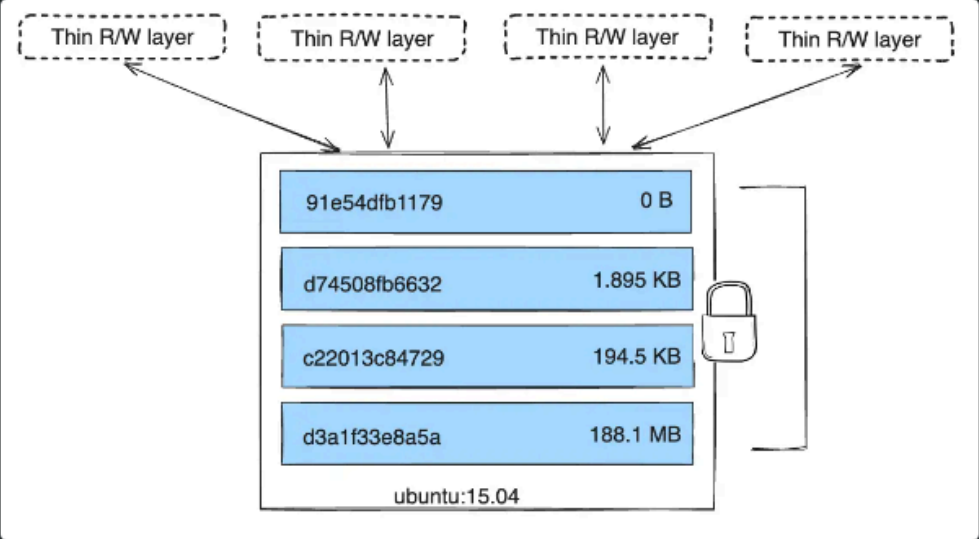
Claim 1	Accused Instrumentalities
<p>applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and</p>	<p>When a Docker image is used to create a container, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications running in their respective containers. The docker image includes essential system files, libraries, and dependencies required to run the software application within the container. The Docker containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it operates in isolation and runs its own instance of the software application without sharing resources or critical system elements with other containers. This ensures that each container has its own isolated context. Docker containers can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same or a different function.</p> <p><i>See, e.g.:</i></p> <p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="674 332 1881 456">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p data-bbox="653 1117 1218 1149">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p>



Claim 1	Accused Instrumentalities
<p>[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p>	<p>In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p> <p>For example, In Docker, each container operates independently, and a Docker base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When a Docker image is used to create a container in ECS, an instance of the SLCSE is provided to a software application. Therefore, different instances of the SLCSE are provided to different applications for performing either a same or a different function, simultaneously.</p> <p><i>See, e.g.:</i></p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities										
	<p data-bbox="674 334 1881 456">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p> <div data-bbox="793 516 1766 1052"><table border="1" data-bbox="1050 665 1501 974"><thead><tr><th>Layer ID</th><th>Size</th></tr></thead><tbody><tr><td>91e54dfb1179</td><td>0 B</td></tr><tr><td>d74508fb6632</td><td>1.895 KB</td></tr><tr><td>c22013c84729</td><td>194.5 KB</td></tr><tr><td>d3a1f33e8a5a</td><td>188.1 MB</td></tr></tbody></table><p data-bbox="1176 998 1312 1023">ubuntu:15.04</p></div> <p data-bbox="653 1117 1218 1149">https://docs.docker.com/storage/storagedriver/</p> <p data-bbox="659 1192 1755 1330">A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.</p>	Layer ID	Size	91e54dfb1179	0 B	d74508fb6632	1.895 KB	c22013c84729	194.5 KB	d3a1f33e8a5a	188.1 MB
Layer ID	Size										
91e54dfb1179	0 B										
d74508fb6632	1.895 KB										
c22013c84729	194.5 KB										
d3a1f33e8a5a	188.1 MB										

Claim 1	Accused Instrumentalities
	https://docs.docker.com/get-started/overview/

CIVIL COVER SHEET

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. (SEE INSTRUCTIONS ON NEXT PAGE OF THIS FORM.)

I. (a) PLAINTIFFS

VIRTAMOVE, CORP.

(b) County of Residence of First Listed Plaintiff _____
(EXCEPT IN U.S. PLAINTIFF CASES)

(c) Attorneys (Firm Name, Address, and Telephone Number)

Russ August & Kabat, 12424 Wilshire Blvd., 12th Floor,
LA, CA 90025; (310) 826-7474

DEFENDANTS

INTERNATIONAL BUSINESS MACHINES CORP.

County of Residence of First Listed Defendant _____
(IN U.S. PLAINTIFF CASES ONLY)

NOTE: IN LAND CONDEMNATION CASES, USE THE LOCATION OF
THE TRACT OF LAND INVOLVED.

Attorneys (If Known)

II. BASIS OF JURISDICTION (Place an "X" in One Box Only)

- ☐ 1 U.S. Government Plaintiff ☒ 3 Federal Question
(U.S. Government Not a Party)
- ☐ 2 U.S. Government Defendant ☐ 4 Diversity
(Indicate Citizenship of Parties in Item III)

III. CITIZENSHIP OF PRINCIPAL PARTIES (Place an "X" in One Box for Plaintiff and One Box for Defendant)

- | | PTF | DEF | | PTF | DEF |
|---|----------------------------|----------------------------|---|----------------------------|----------------------------|
| Citizen of This State | <input type="checkbox"/> 1 | <input type="checkbox"/> 1 | Incorporated or Principal Place of Business In This State | <input type="checkbox"/> 4 | <input type="checkbox"/> 4 |
| Citizen of Another State | <input type="checkbox"/> 2 | <input type="checkbox"/> 2 | Incorporated and Principal Place of Business In Another State | <input type="checkbox"/> 5 | <input type="checkbox"/> 5 |
| Citizen or Subject of a Foreign Country | <input type="checkbox"/> 3 | <input type="checkbox"/> 3 | Foreign Nation | <input type="checkbox"/> 6 | <input type="checkbox"/> 6 |

IV. NATURE OF SUIT (Place an "X" in One Box Only)Click here for: [Nature of Suit Code Descriptions.](#)

CONTRACT	TORTS	FORFEITURE/PENALTY	BANKRUPTCY	OTHER STATUTES
<input type="checkbox"/> 110 Insurance <input type="checkbox"/> 120 Marine <input type="checkbox"/> 130 Miller Act <input type="checkbox"/> 140 Negotiable Instrument <input type="checkbox"/> 150 Recovery of Overpayment & Enforcement of Judgment <input type="checkbox"/> 151 Medicare Act <input type="checkbox"/> 152 Recovery of Defaulted Student Loans (Excludes Veterans) <input type="checkbox"/> 153 Recovery of Overpayment of Veteran's Benefits <input type="checkbox"/> 160 Stockholders' Suits <input type="checkbox"/> 190 Other Contract <input type="checkbox"/> 195 Contract Product Liability <input type="checkbox"/> 196 Franchise	PERSONAL INJURY <input type="checkbox"/> 310 Airplane <input type="checkbox"/> 315 Airplane Product Liability <input type="checkbox"/> 320 Assault, Libel & Slander <input type="checkbox"/> 330 Federal Employers' Liability <input type="checkbox"/> 340 Marine <input type="checkbox"/> 345 Marine Product Liability <input type="checkbox"/> 350 Motor Vehicle <input type="checkbox"/> 355 Motor Vehicle Product Liability <input type="checkbox"/> 360 Other Personal Injury <input type="checkbox"/> 362 Personal Injury - Medical Malpractice PERSONAL INJURY <input type="checkbox"/> 365 Personal Injury - Product Liability <input type="checkbox"/> 367 Health Care/Pharmaceutical Personal Injury Product Liability <input type="checkbox"/> 368 Asbestos Personal Injury Product Liability PERSONAL PROPERTY <input type="checkbox"/> 370 Other Fraud <input type="checkbox"/> 371 Truth in Lending <input type="checkbox"/> 380 Other Personal Property Damage <input type="checkbox"/> 385 Property Damage Product Liability	<input type="checkbox"/> 625 Drug Related Seizure of Property 21 USC 881 <input type="checkbox"/> 690 Other LABOR <input type="checkbox"/> 710 Fair Labor Standards Act <input type="checkbox"/> 720 Labor/Management Relations <input type="checkbox"/> 740 Railway Labor Act <input type="checkbox"/> 751 Family and Medical Leave Act <input type="checkbox"/> 790 Other Labor Litigation <input type="checkbox"/> 791 Employee Retirement Income Security Act IMMIGRATION <input type="checkbox"/> 462 Naturalization Application <input type="checkbox"/> 465 Other Immigration Actions	<input type="checkbox"/> 422 Appeal 28 USC 158 <input type="checkbox"/> 423 Withdrawal 28 USC 157 PROPERTY RIGHTS <input checked="" type="checkbox"/> 820 Copyrights <input type="checkbox"/> 830 Patent <input type="checkbox"/> 835 Patent - Abbreviated New Drug Application <input type="checkbox"/> 840 Trademark <input type="checkbox"/> 880 Defend Trade Secrets Act of 2016 SOCIAL SECURITY <input type="checkbox"/> 861 HIA (1395ff) <input type="checkbox"/> 862 Black Lung (923) <input type="checkbox"/> 863 DIWC/DIWW (405(g)) <input type="checkbox"/> 864 SSID Title XVI <input type="checkbox"/> 865 RSI (405(g)) FEDERAL TAX SUITS <input type="checkbox"/> 870 Taxes (U.S. Plaintiff or Defendant) <input type="checkbox"/> 871 IRS—Third Party 26 USC 7609	<input type="checkbox"/> 375 False Claims Act <input type="checkbox"/> 376 Qui Tam (31 USC 3729(a)) <input type="checkbox"/> 400 State Reapportionment <input type="checkbox"/> 410 Antitrust <input type="checkbox"/> 430 Banks and Banking <input type="checkbox"/> 450 Commerce <input type="checkbox"/> 460 Deportation <input type="checkbox"/> 470 Racketeer Influenced and Corrupt Organizations <input type="checkbox"/> 480 Consumer Credit (15 USC 1681 or 1692) <input type="checkbox"/> 485 Telephone Consumer Protection Act <input type="checkbox"/> 490 Cable/Sat TV <input type="checkbox"/> 850 Securities/Commodities/Exchange <input type="checkbox"/> 890 Other Statutory Actions <input type="checkbox"/> 891 Agricultural Acts <input type="checkbox"/> 893 Environmental Matters <input type="checkbox"/> 895 Freedom of Information Act <input type="checkbox"/> 896 Arbitration <input type="checkbox"/> 899 Administrative Procedure Act/Review or Appeal of Agency Decision <input type="checkbox"/> 950 Constitutionality of State Statutes
REAL PROPERTY <input type="checkbox"/> 210 Land Condemnation <input type="checkbox"/> 220 Foreclosure <input type="checkbox"/> 230 Rent Lease & Ejectment <input type="checkbox"/> 240 Torts to Land <input type="checkbox"/> 245 Tort Product Liability <input type="checkbox"/> 290 All Other Real Property	CIVIL RIGHTS <input type="checkbox"/> 440 Other Civil Rights <input type="checkbox"/> 441 Voting <input type="checkbox"/> 442 Employment <input type="checkbox"/> 443 Housing/Accommodations <input type="checkbox"/> 445 Amer. w/Disabilities - Employment <input type="checkbox"/> 446 Amer. w/Disabilities - Other <input type="checkbox"/> 448 Education PRISONER PETITIONS Habeas Corpus: <input type="checkbox"/> 463 Alien Detainee <input type="checkbox"/> 510 Motions to Vacate Sentence <input type="checkbox"/> 530 General <input type="checkbox"/> 535 Death Penalty Other: <input type="checkbox"/> 540 Mandamus & Other <input type="checkbox"/> 550 Civil Rights <input type="checkbox"/> 555 Prison Condition <input type="checkbox"/> 560 Civil Detainee - Conditions of Confinement			

V. ORIGIN (Place an "X" in One Box Only)

- ☒ 1 Original Proceeding ☐ 2 Removed from State Court ☐ 3 Remanded from Appellate Court ☐ 4 Reinstated or Reopened ☐ 5 Transferred from Another District (specify) ☐ 6 Multidistrict Litigation - Transfer ☐ 8 Multidistrict Litigation - Direct File

VI. CAUSE OF ACTION

Cite the U.S. Civil Statute under which you are filing (Do not cite jurisdictional statutes unless diversity):
35 U.S.C. § 1 et seq.

Brief description of cause:
Patent Infringement

VII. REQUESTED IN COMPLAINT:

☐ CHECK IF THIS IS A CLASS ACTION UNDER RULE 23, F.R.Cv.P.

DEMAND \$

CHECK YES only if demanded in complaint:

JURY DEMAND: ☒ Yes ☐ No**VIII. RELATED CASE(S) IF ANY**

(See instructions):

JUDGE _____

DOCKET NUMBER _____

DATE

01/31/2024

SIGNATURE OF ATTORNEY OF RECORD

/s/ Reza Mirzaie

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG. JUDGE _____

INSTRUCTIONS FOR ATTORNEYS COMPLETING CIVIL COVER SHEET FORM JS 44

Authority For Civil Cover Sheet

The JS 44 civil cover sheet and the information contained herein neither replaces nor supplements the filings and service of pleading or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. Consequently, a civil cover sheet is submitted to the Clerk of Court for each civil complaint filed. The attorney filing a case should complete the form as follows:

- I.(a) Plaintiffs-Defendants.** Enter names (last, first, middle initial) of plaintiff and defendant. If the plaintiff or defendant is a government agency, use only the full name or standard abbreviations. If the plaintiff or defendant is an official within a government agency, identify first the agency and then the official, giving both name and title.
- (b) County of Residence.** For each civil case filed, except U.S. plaintiff cases, enter the name of the county where the first listed plaintiff resides at the time of filing. In U.S. plaintiff cases, enter the name of the county in which the first listed defendant resides at the time of filing. (NOTE: In land condemnation cases, the county of residence of the "defendant" is the location of the tract of land involved.)
- (c) Attorneys.** Enter the firm name, address, telephone number, and attorney of record. If there are several attorneys, list them on an attachment, noting in this section "(see attachment)".
- II. Jurisdiction.** The basis of jurisdiction is set forth under Rule 8(a), F.R.Cv.P., which requires that jurisdictions be shown in pleadings. Place an "X" in one of the boxes. If there is more than one basis of jurisdiction, precedence is given in the order shown below.
 United States plaintiff. (1) Jurisdiction based on 28 U.S.C. 1345 and 1348. Suits by agencies and officers of the United States are included here.
 United States defendant. (2) When the plaintiff is suing the United States, its officers or agencies, place an "X" in this box.
 Federal question. (3) This refers to suits under 28 U.S.C. 1331, where jurisdiction arises under the Constitution of the United States, an amendment to the Constitution, an act of Congress or a treaty of the United States. In cases where the U.S. is a party, the U.S. plaintiff or defendant code takes precedence, and box 1 or 2 should be marked.
 Diversity of citizenship. (4) This refers to suits under 28 U.S.C. 1332, where parties are citizens of different states. When Box 4 is checked, the citizenship of the different parties must be checked. (See Section III below; **NOTE: federal question actions take precedence over diversity cases.**)
- III. Residence (citizenship) of Principal Parties.** This section of the JS 44 is to be completed if diversity of citizenship was indicated above. Mark this section for each principal party.
- IV. Nature of Suit.** Place an "X" in the appropriate box. If there are multiple nature of suit codes associated with the case, pick the nature of suit code that is most applicable. Click here for: [Nature of Suit Code Descriptions](#).
- V. Origin.** Place an "X" in one of the seven boxes.
 Original Proceedings. (1) Cases which originate in the United States district courts.
 Removed from State Court. (2) Proceedings initiated in state courts may be removed to the district courts under Title 28 U.S.C., Section 1441.
 Remanded from Appellate Court. (3) Check this box for cases remanded to the district court for further action. Use the date of remand as the filing date.
 Reinstated or Reopened. (4) Check this box for cases reinstated or reopened in the district court. Use the reopening date as the filing date.
 Transferred from Another District. (5) For cases transferred under Title 28 U.S.C. Section 1404(a). Do not use this for within district transfers or multidistrict litigation transfers.
 Multidistrict Litigation – Transfer. (6) Check this box when a multidistrict case is transferred into the district under authority of Title 28 U.S.C. Section 1407.
 Multidistrict Litigation – Direct File. (8) Check this box when a multidistrict case is filed in the same district as the Master MDL docket.
PLEASE NOTE THAT THERE IS NOT AN ORIGIN CODE 7. Origin Code 7 was used for historical records and is no longer relevant due to changes in statute.
- VI. Cause of Action.** Report the civil statute directly related to the cause of action and give a brief description of the cause. **Do not cite jurisdictional statutes unless diversity.** Example: U.S. Civil Statute: 47 USC 553 Brief Description: Unauthorized reception of cable service.
- VII. Requested in Complaint.** Class Action. Place an "X" in this box if you are filing a class action under Rule 23, F.R.Cv.P.
 Demand. In this space enter the actual dollar amount being demanded or indicate other demand, such as a preliminary injunction.
 Jury Demand. Check the appropriate box to indicate whether or not a jury is being demanded.
- VIII. Related Cases.** This section of the JS 44 is used to reference related pending cases, if any. If there are related pending cases, insert the docket numbers and the corresponding judge names for such cases.

Date and Attorney Signature. Date and sign the civil cover sheet.

EXHIBIT J



US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,519,814 B2**

(45) **Date of Patent:** **Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

WO WO 2006/039181 A 4/2006

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

OTHER PUBLICATIONS

Soltesz, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, <http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667>.*

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

Primary Examiner—Kambiz Zand

Assistant Examiner—Ronald Baum

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248;
709/214; 719/319

(58) **Field of Classification Search** 713/167;
719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

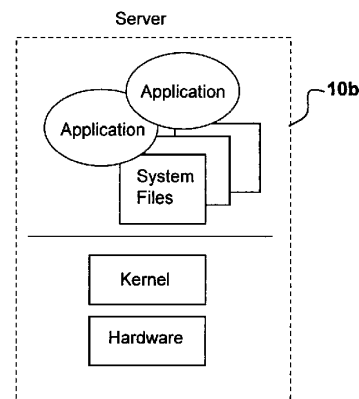
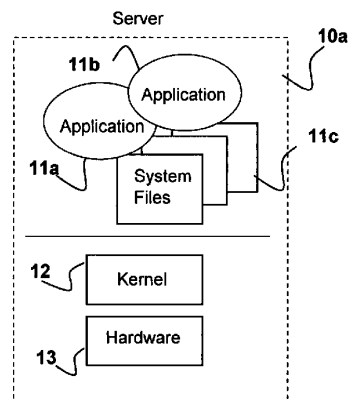
FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



US 7,519,814 B2

Page 2

U.S. PATENT DOCUMENTS							
6,847,970	B2 *	1/2005	Keller et al.	707/100	2002/0004854	A1	1/2002 Hartley
7,076,784	B1 *	7/2006	Russell et al.	719/315	2002/0174215	A1	11/2002 Schaefer 709/224
7,287,259	B2 *	10/2007	Grier et al.	719/331	2003/0101292	A1	5/2003 Fisher
					* cited by examiner		

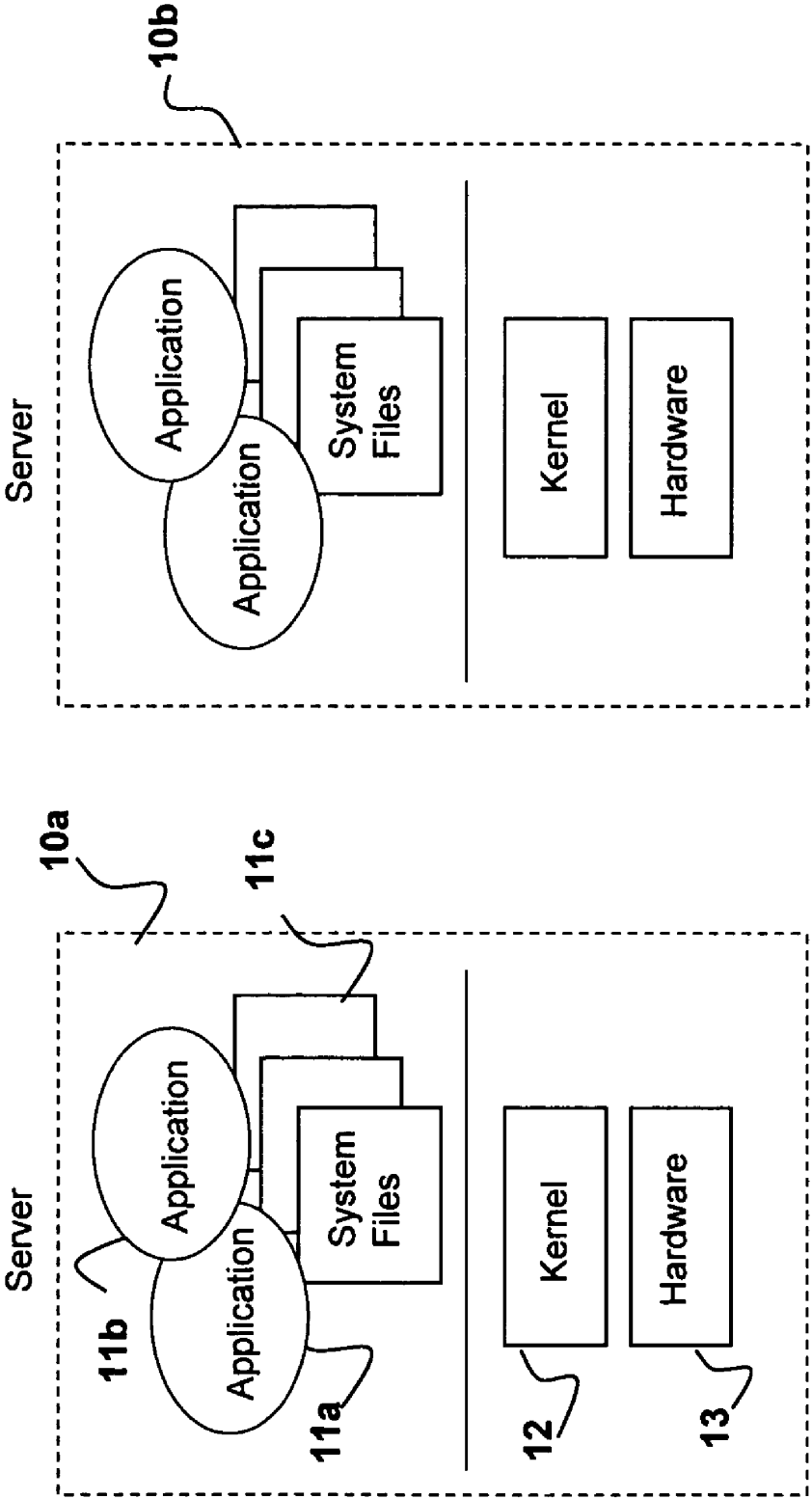


Figure 1

U.S. Patent

Apr. 14, 2009

Sheet 2 of 17

US 7,519,814 B2

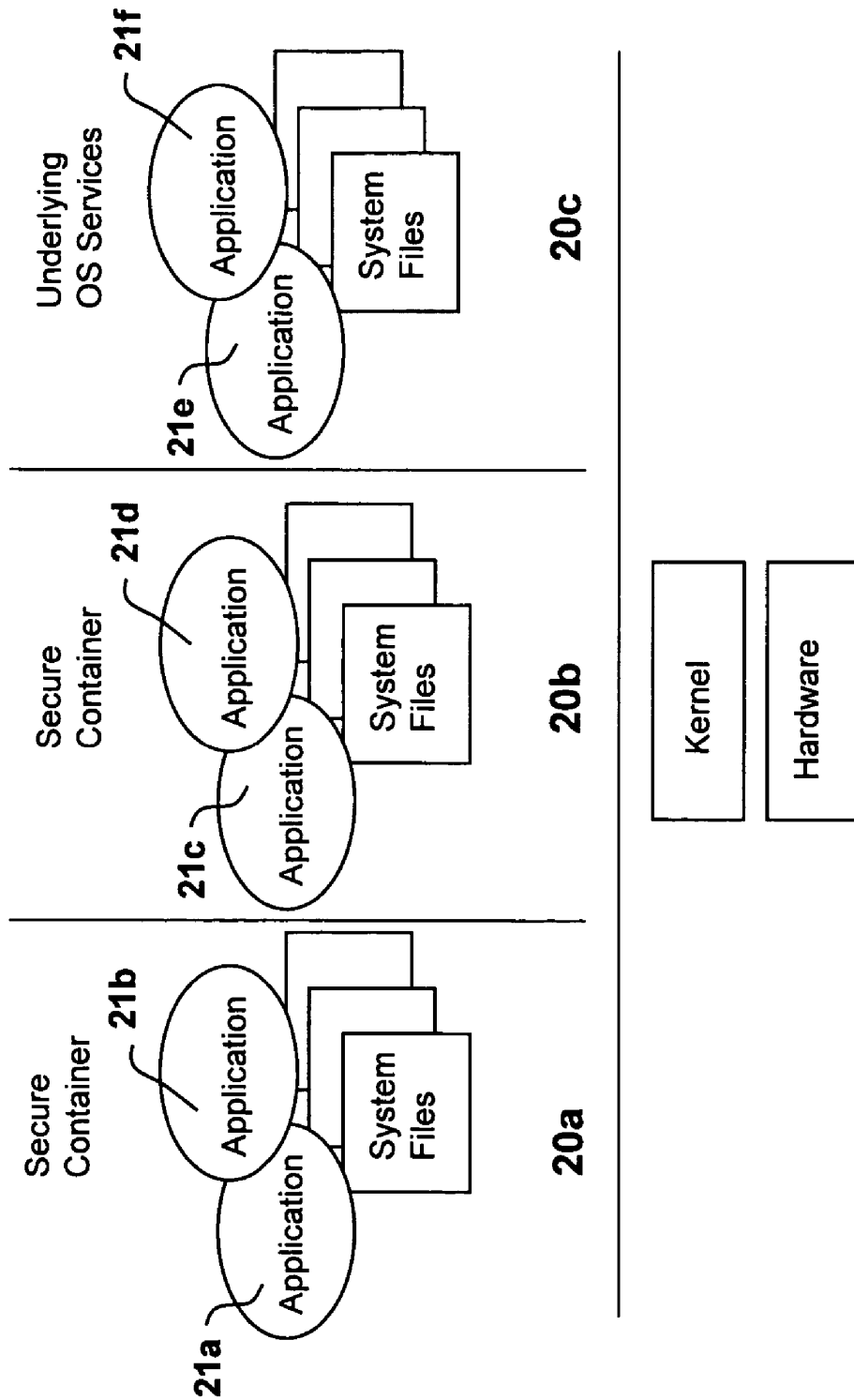


Figure 2

U.S. Patent

Apr. 14, 2009

Sheet 3 of 17

US 7,519,814 B2

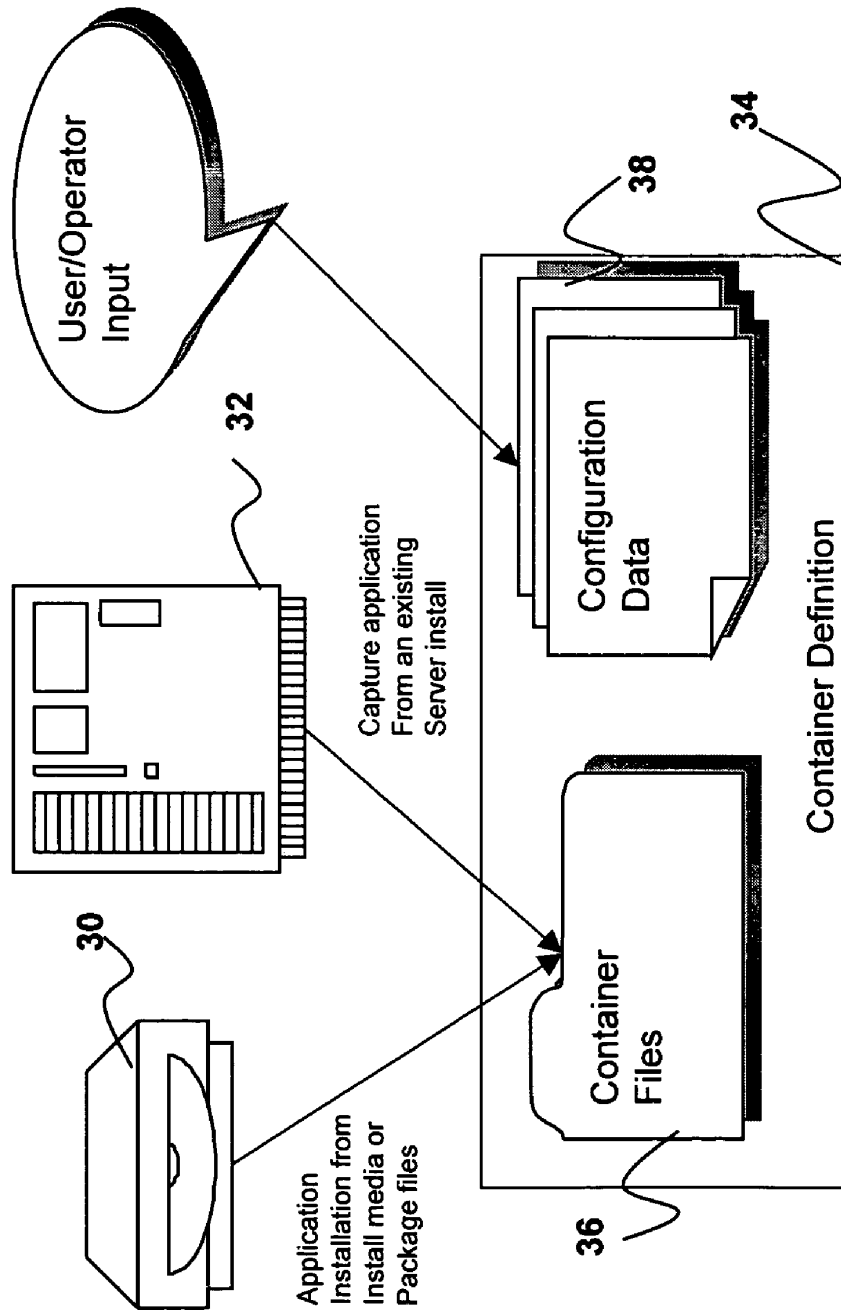


Figure 3

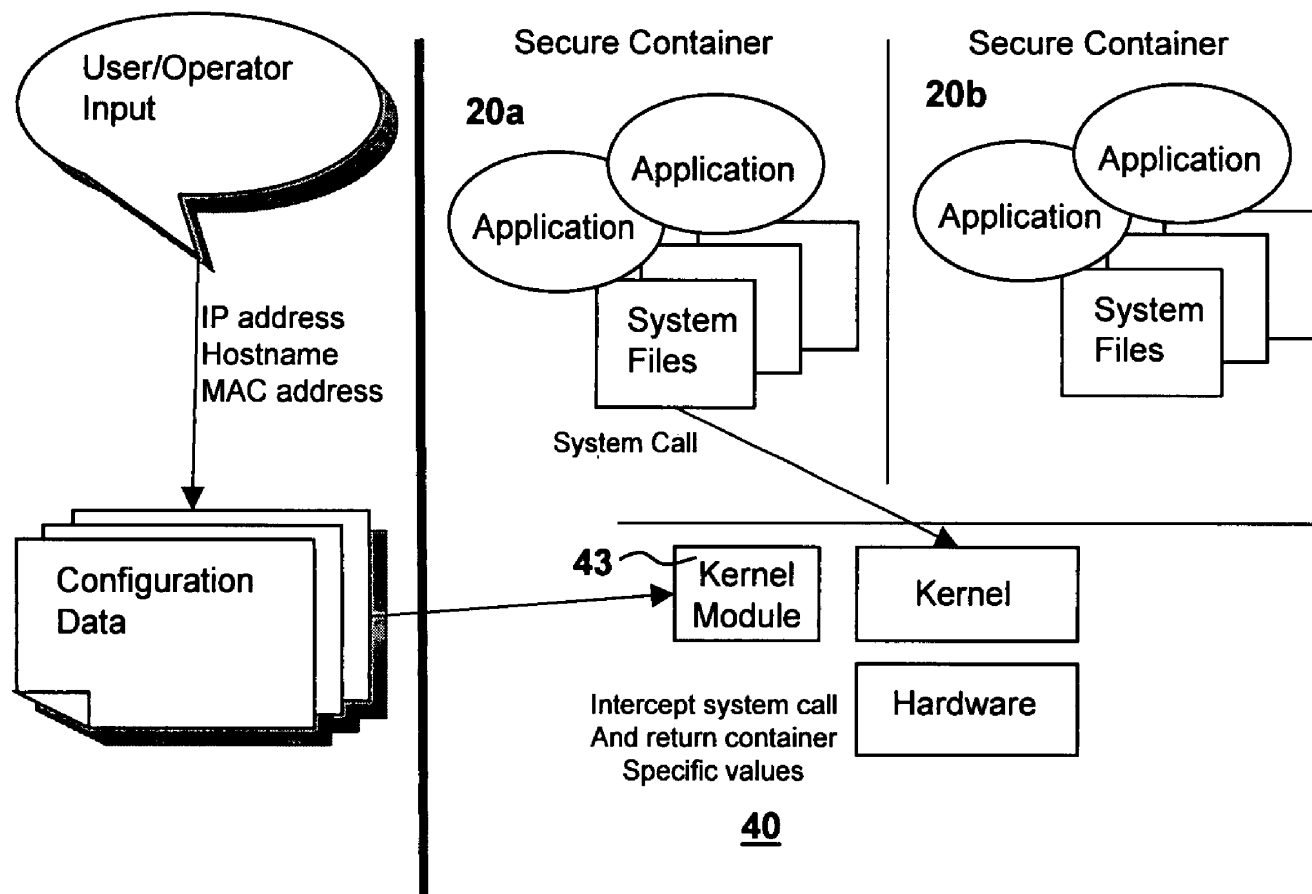


Figure 4

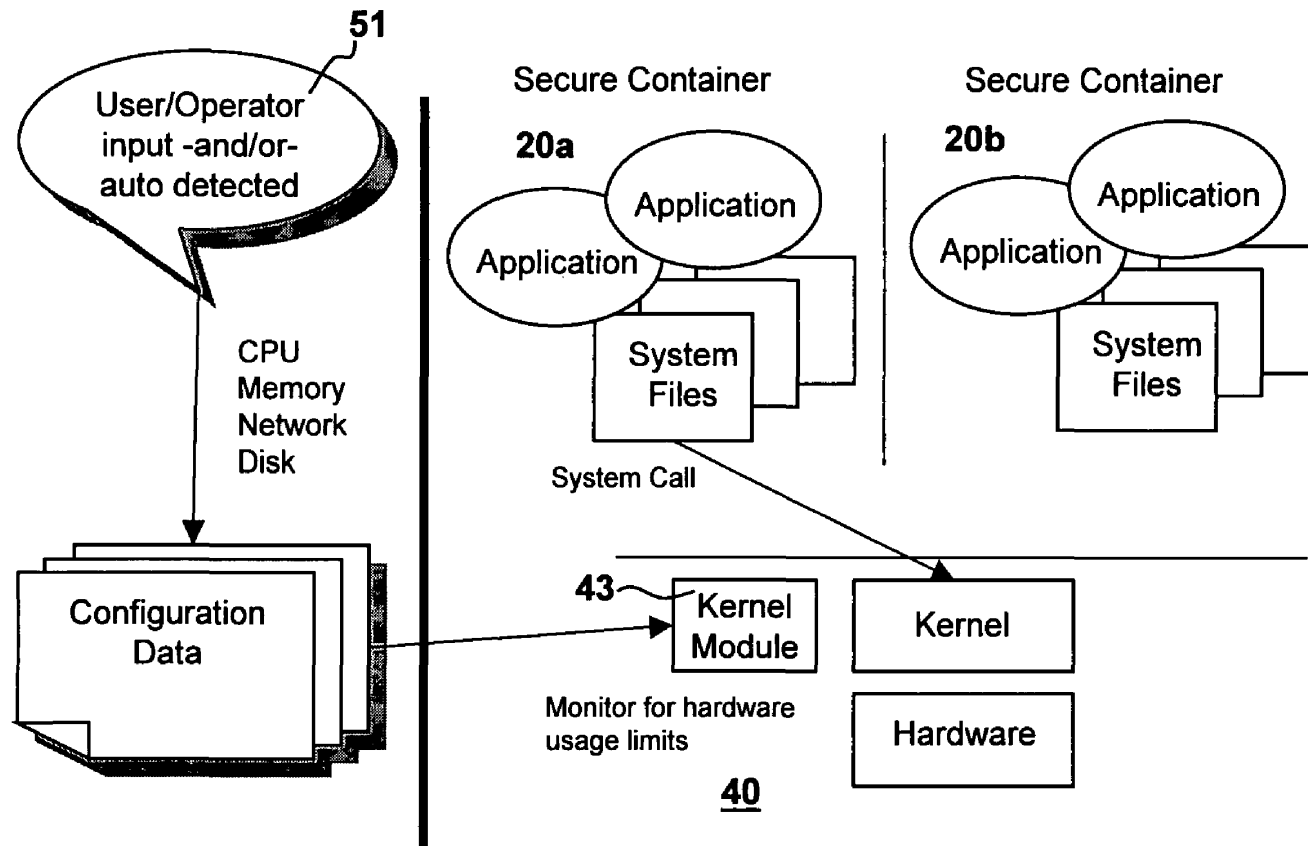


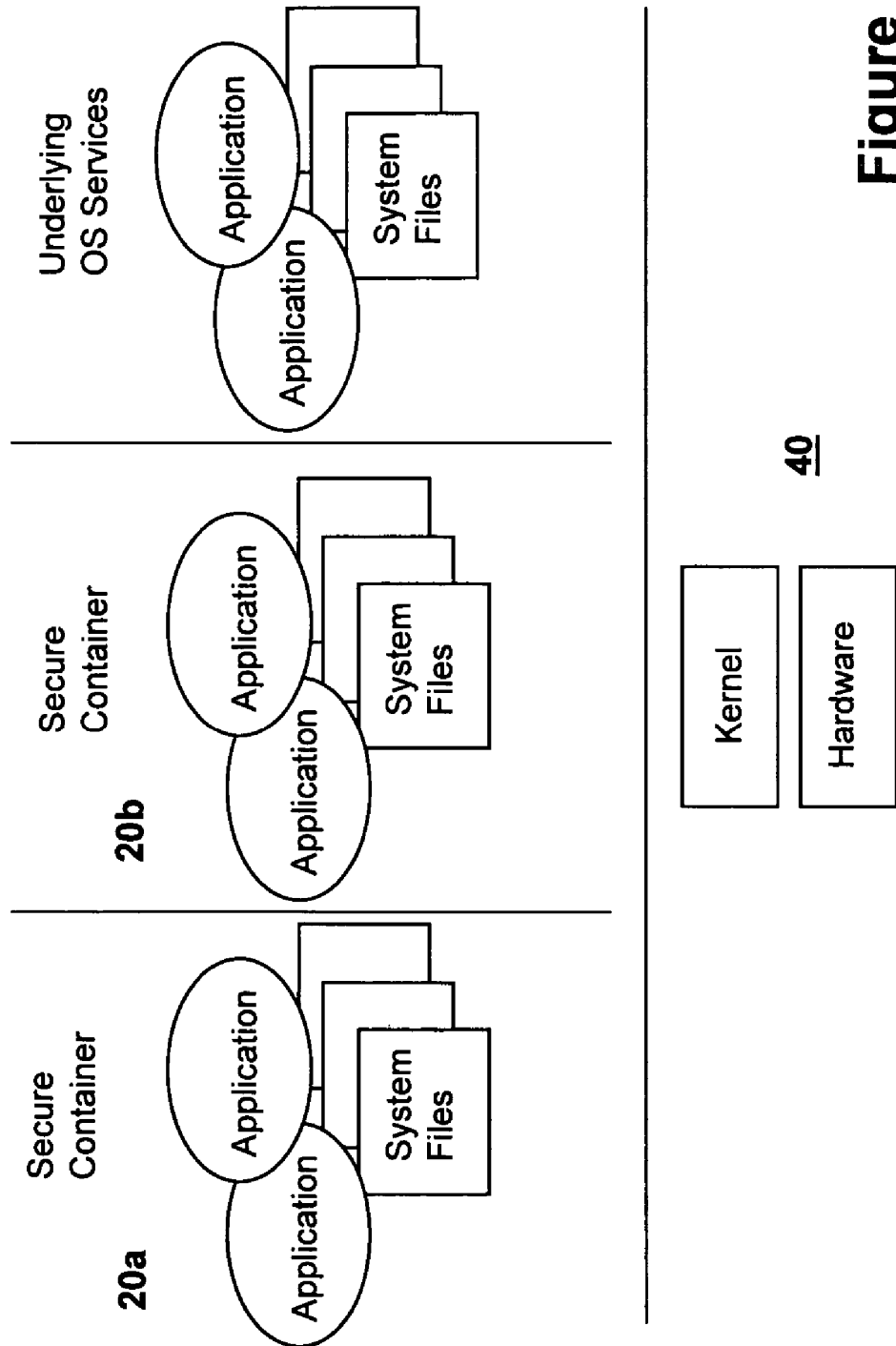
Figure 5

U.S. Patent

Apr. 14, 2009

Sheet 6 of 17

US 7,519,814 B2



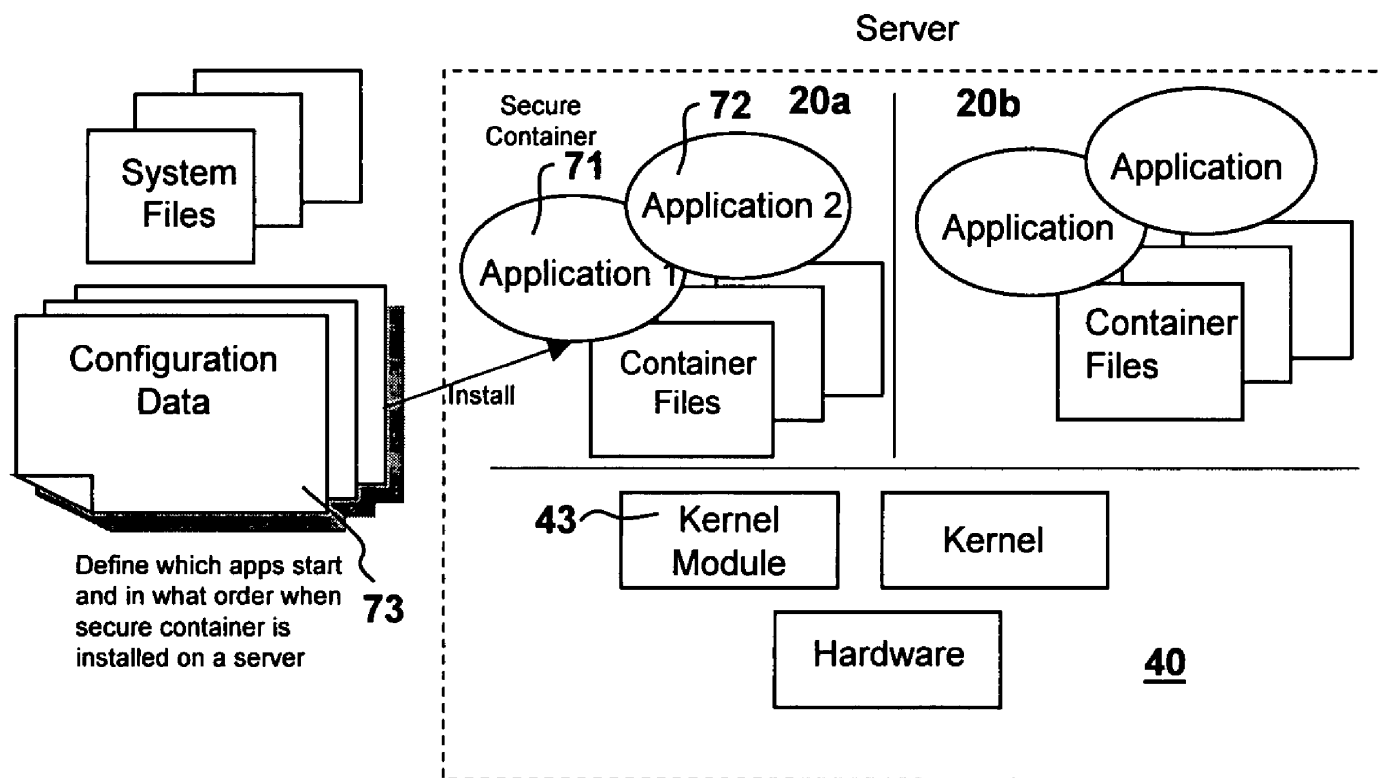


Figure 7

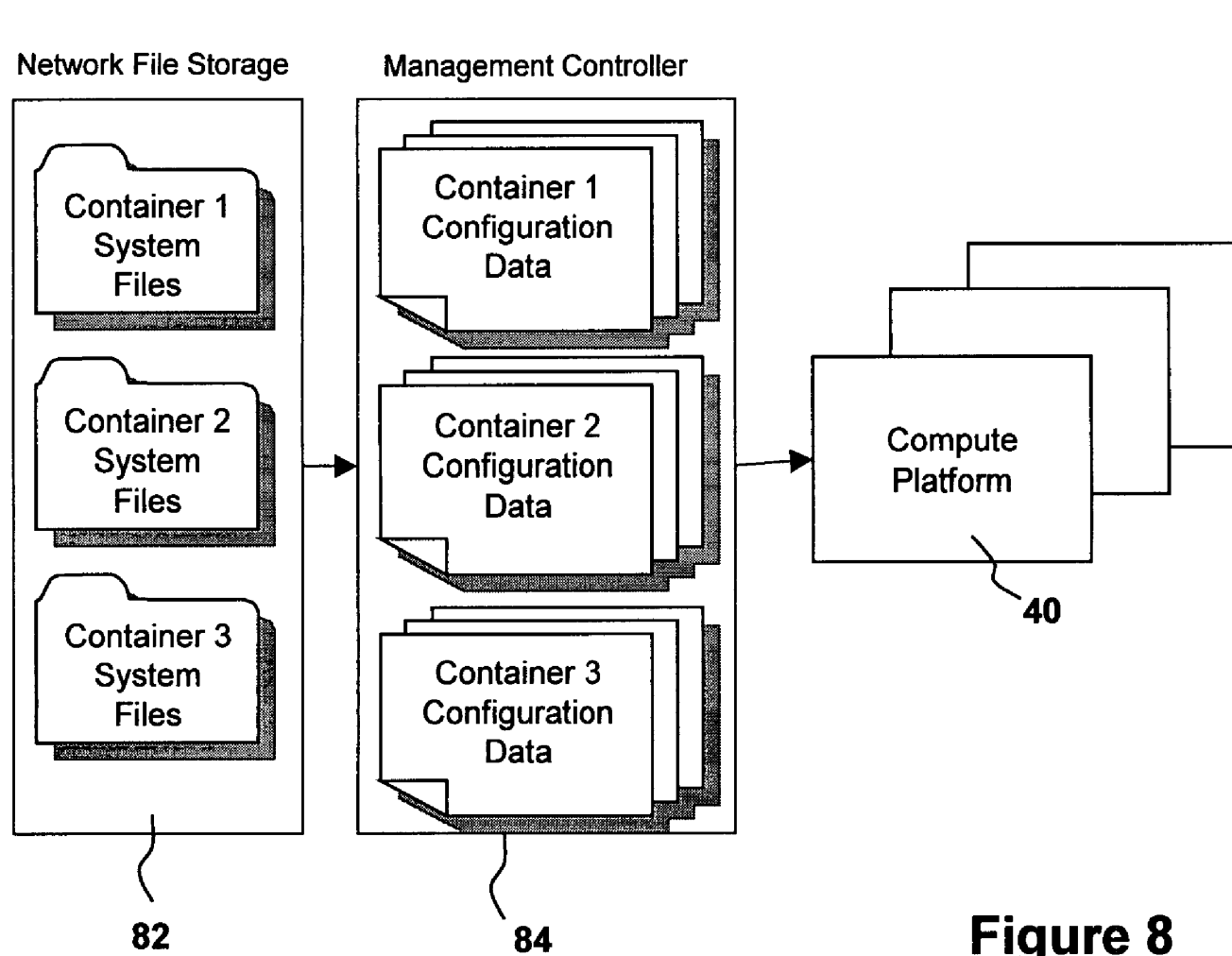


Figure 8

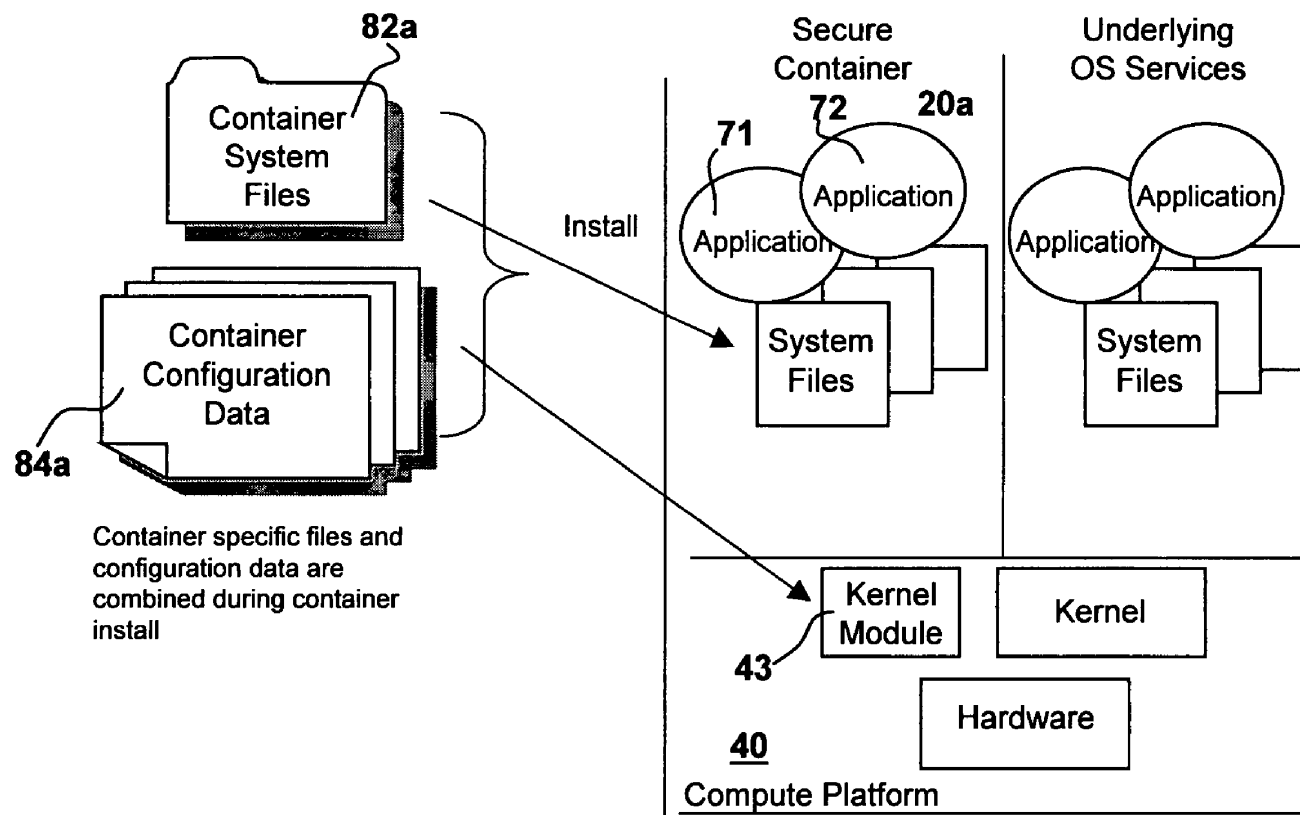


Figure 9

U.S. Patent

Apr. 14, 2009

Sheet 10 of 17

US 7,519,814 B2

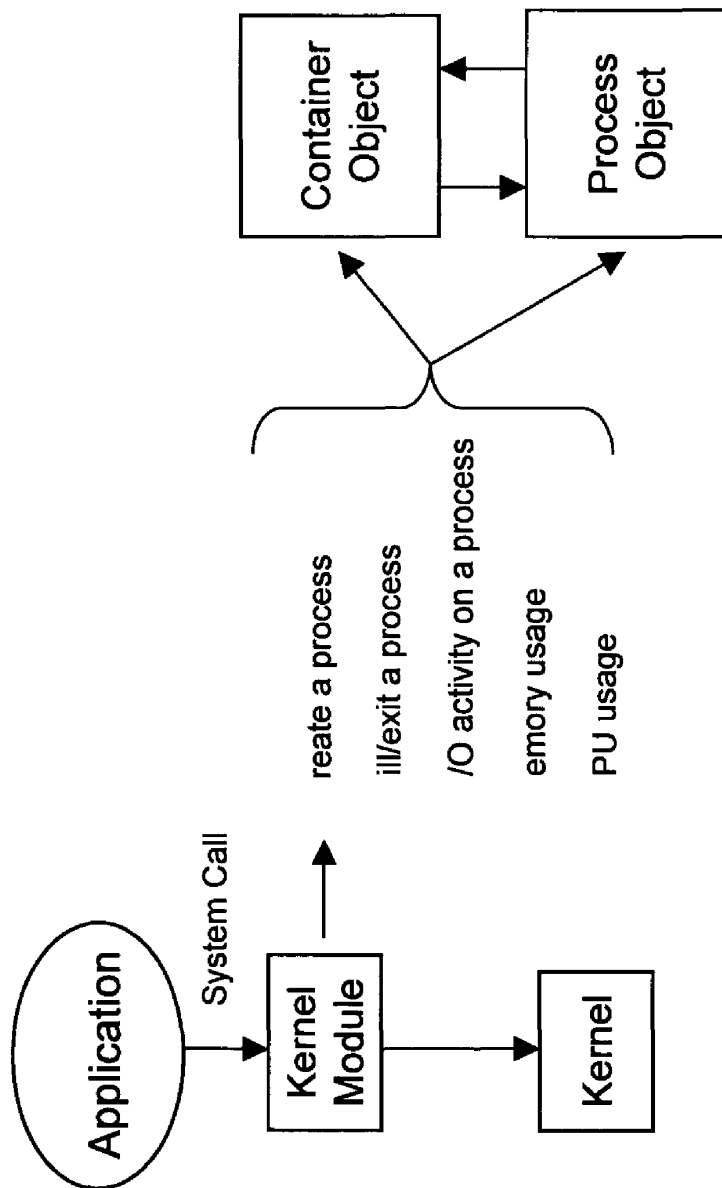


Figure 10

U.S. Patent

Apr. 14, 2009

Sheet 11 of 17

US 7,519,814 B2

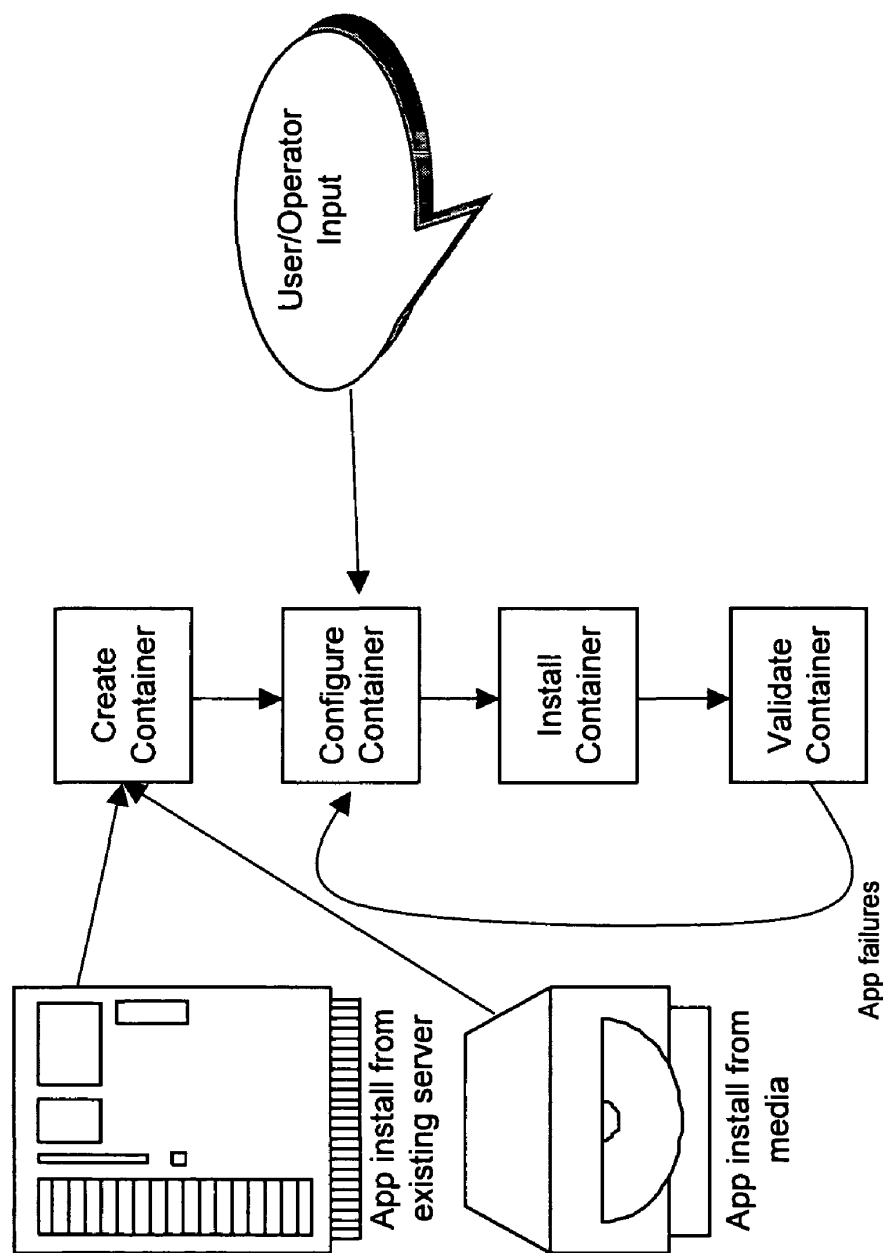


Figure 11

U.S. Patent

Apr. 14, 2009

Sheet 12 of 17

US 7,519,814 B2

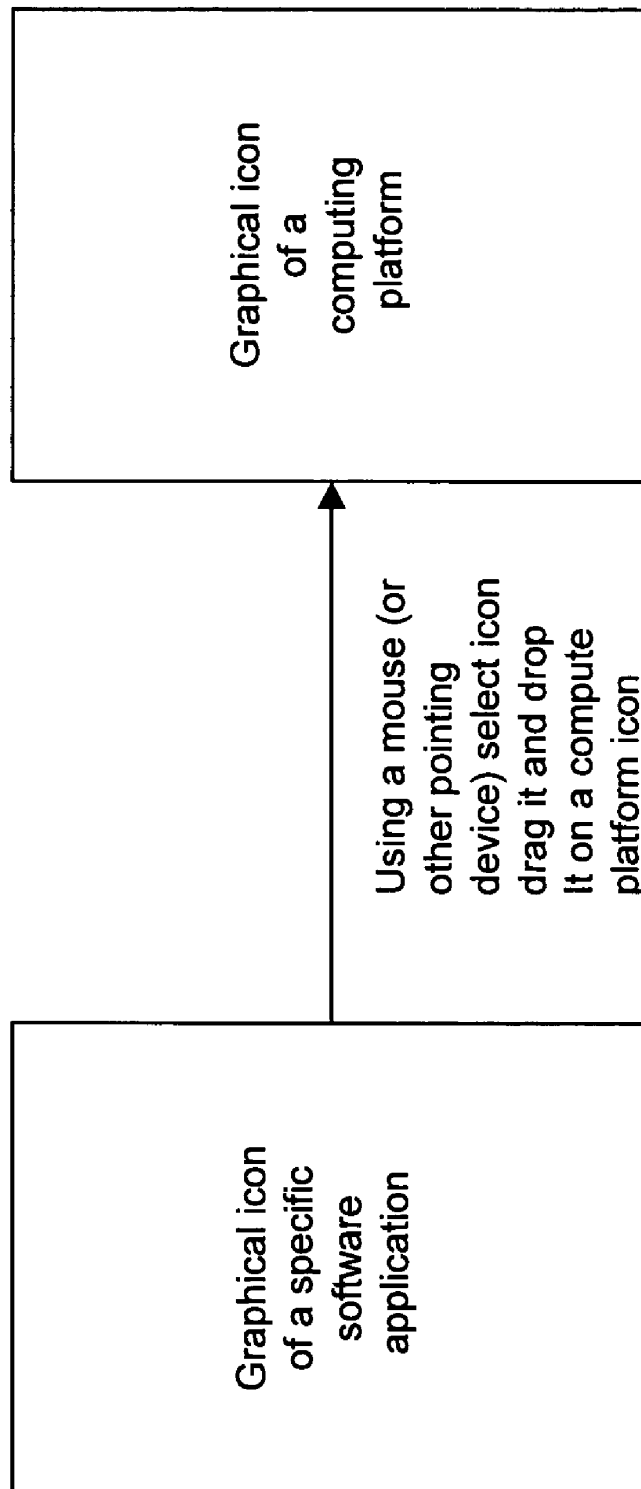


Figure 12

U.S. Patent

Apr. 14, 2009

Sheet 13 of 17

US 7,519,814 B2

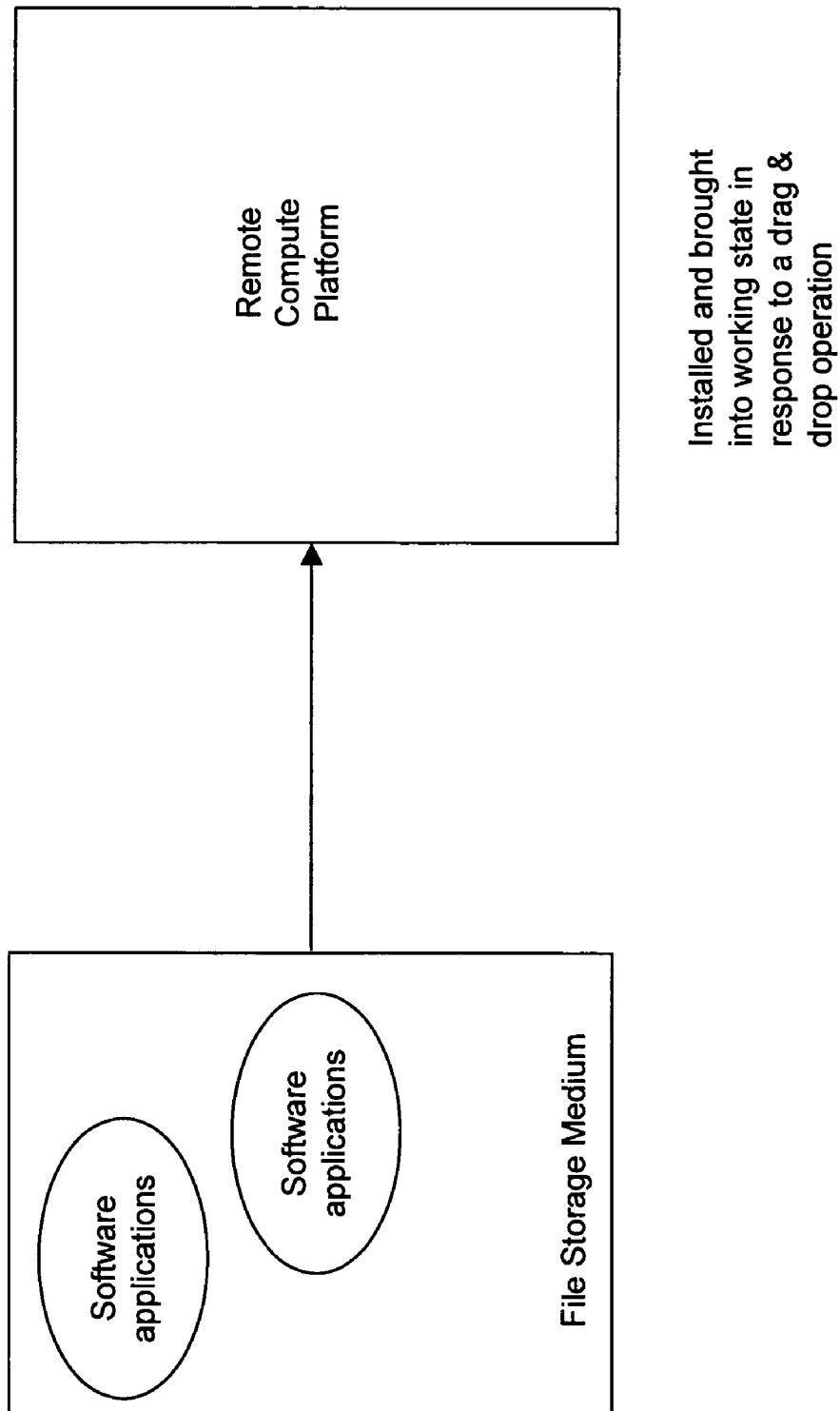


Figure 13

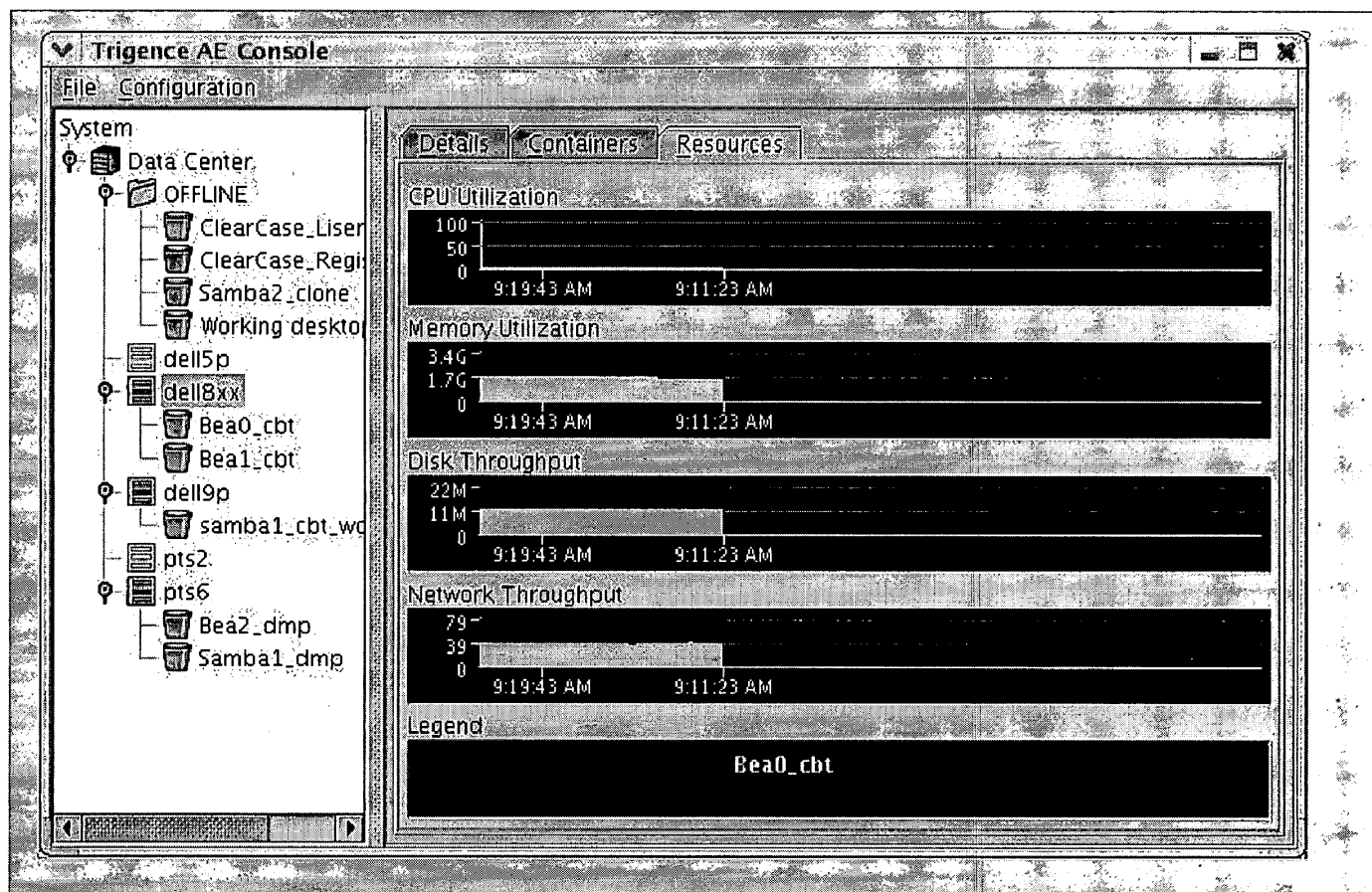


Figure 14

U.S. Patent

Apr. 14, 2009

Sheet 14 of 17

US 7,519,814 B2

U.S. Patent

Apr. 14, 2009

Sheet 15 of 17

US 7,519,814 B2

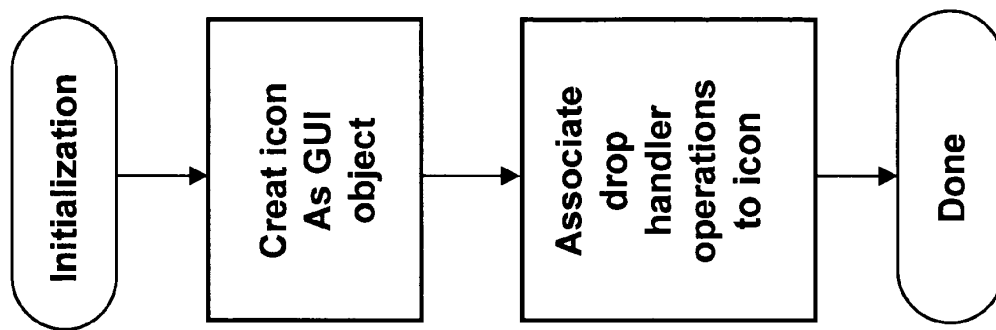


Figure 15

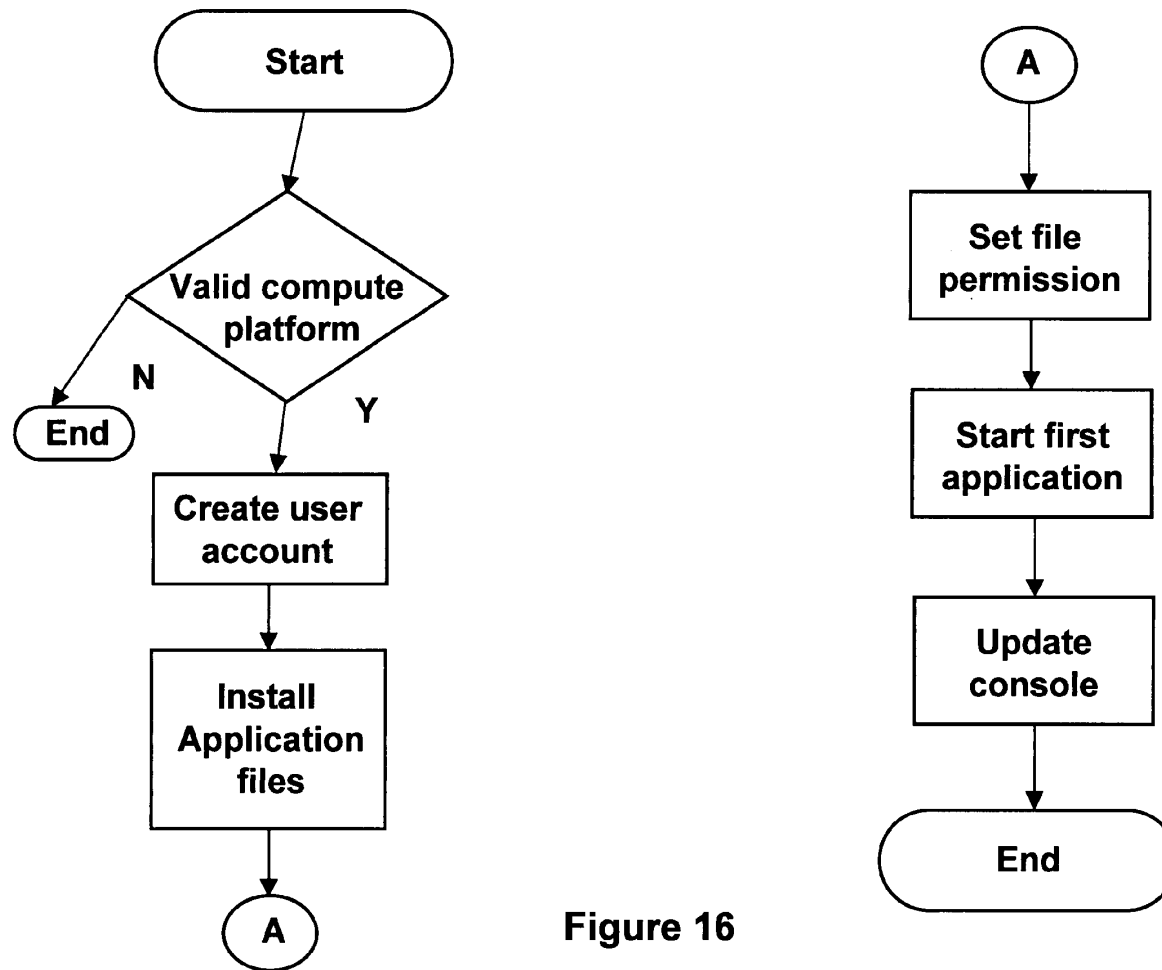


Figure 16

U.S. Patent

Apr. 14, 2009

Sheet 17 of 17

US 7,519,814 B2

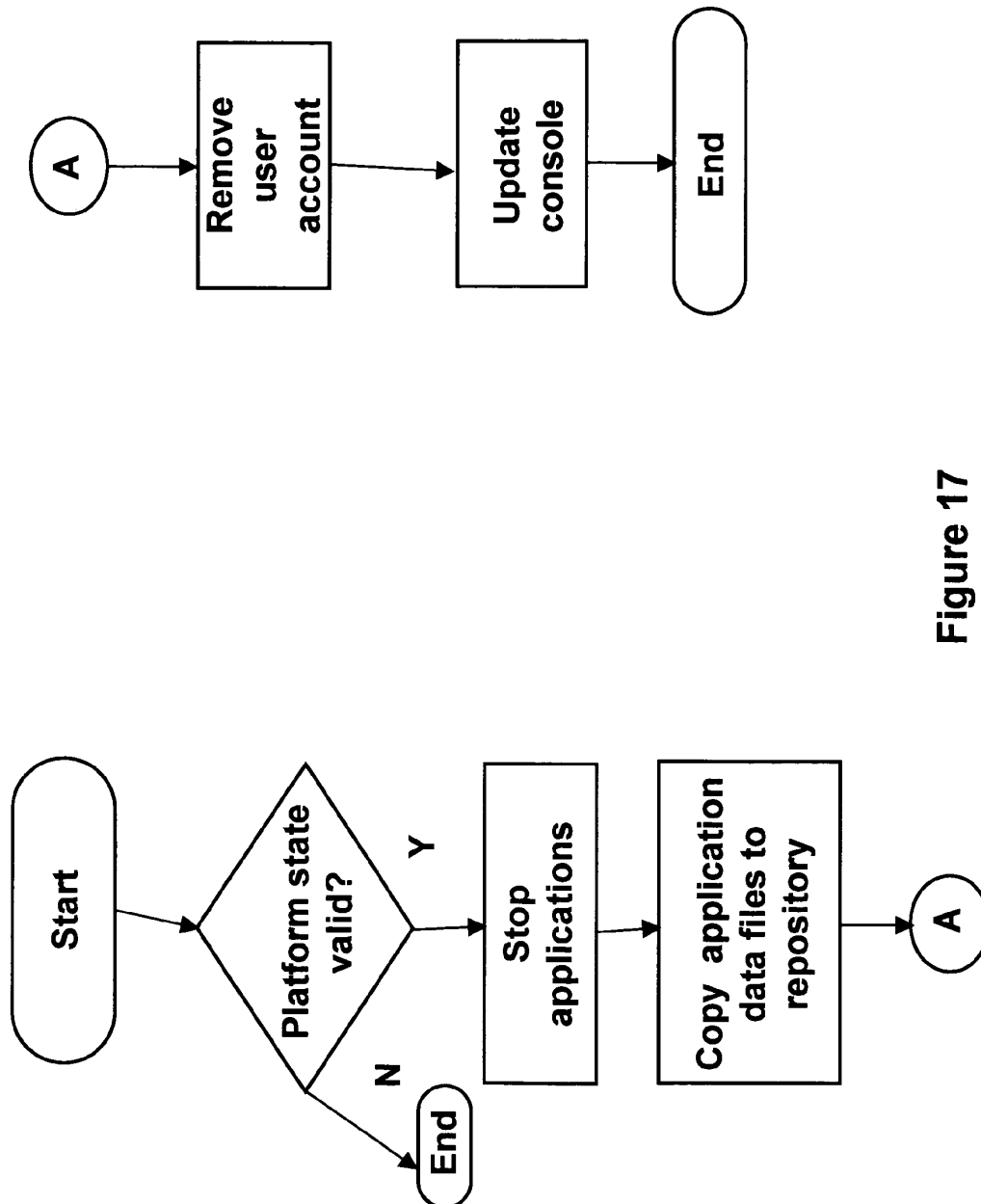


Figure 17

US 7,519,814 B2

1

**SYSTEM FOR CONTAINERIZATION OF
APPLICATION SETS****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

2

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

```
/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1
```

US 7,519,814 B2

3

/lib/libnsl.so.1
 /lib/libdl.so.2
 /lib/i686/libpthread.so.0
 /lib/i686/libc.so.6
 /lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
 /etc/httpd/conf
 /etc/httpd/conf.d
 /etc/httpd/logs
 /etc/httpd/modules
 /etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

4

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySQL and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

US 7,519,814 B2

5

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of

6

this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed “spoofing” and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. 3 is a pictorial diagram illustrating the creation of a container.

FIG. 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the “kernel module” which is used to handle system calls.

FIG. 5 is a diagram similar to FIG. 4, wherein additional configuration data is provided.

FIG. 6 is a diagram illustrating a system wherein the containers are secure containers.

FIG. 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. 9 is a diagram that illustrates the installation of a container on a server.

FIG. 10 is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. 14 is a screen snapshot of an implementation according to the schematic of FIG. 13.

FIG. 15 is a flow chart of a method of initializing icons of FIG. 14.

FIG. 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. 14; and,

FIG. 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13.

DETAILED DESCRIPTION

Turning now to FIG. 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c.

US 7,519,814 B2

7

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. 2 illustrates a system in accordance with the invention in which multiple applications **21a**, **21b**, **21c**, **21d**, **21e**, and **21f** can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20a**, **20b** and **20c**. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20a**, **20b** and **20c** are shown in FIG. 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. 4 illustrates a system having two secure containers **20a** **20b**, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20a**, **20b** environment system calls are intercepted, by the kernel module **43** passing through services installed as part of

8

this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

As introduced heretofore, “spoofing” is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system’s kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the ‘uname’ system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has ‘spoofed’ the ‘uname’ system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same “spoofing” mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

- tracking applications, for example a tracking of which applications are in and out of a specific container;
- spoofing particular values returned by the kernel;
- applying resource checks and imposing resource limits;
- monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,
- retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20a** or **20b**. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. 6 that each application executing associated with a container **20a** or **20b**, or contained within a

US 7,519,814 B2

9

container **20a** or **20b**, is able to access files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20a** association are not able to access the files contained in another container **20b**.

When a container **20a** or **20b** is installed specific applications are started, as is shown in FIG. 7 and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. 7, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. 8, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. 9 shows that the files **82a**, **84a** are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing

10

the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. 9.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform and then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. 3, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

US 7,519,814 B2

11

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

- modifying the contents a file to add a container specific IP address;
- modifying the contents of a directory to define start scripts that should be executed;
- modifying the contents of a file to define which mount points will relate to a container;
- removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications

12

are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

US 7,519,814 B2

13

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

14

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

US 7,519,814 B2

15

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. 14, a screen snapshot of an implementation is shown according to the schematic of FIG. 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. 15, shown is a flow chart of a method of initializing the icons of FIG. 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. 16 are performed by an install drop handler. During installation, verification is performed to

16

determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. 15), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

US 7,519,814 B2

17

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

18

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

- a) running an instance of a service on a server;
- b) determining which files are being used; and,
- c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim 14 comprising the steps of:

- assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
- installing the container on a server; and,
- testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

- using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with an unique icon representing

US 7,519,814 B2

19

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim 20 further comprising starting a distributed software application.

22. A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim 17 further comprising creating a user account for the service.

25. A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim 1, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;
displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

20

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim 31, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and,

forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

EXHIBIT K



US007784058B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,784,058 B2**

(45) **Date of Patent:** **Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE
 CRITICAL SYSTEM ELEMENTS AS SHARED
 LIBRARIES**

2002/0004854 A1 1/2002 Hartley
 2002/0174215 A1 11/2002 Schaefer 709/224
 2003/0101292 A1 5/2003 Fisher
 2004/0025165 A1* 2/2004 Desoli et al. 719/310

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul
 O'Leary**, Kanata (CA); **Dean Huffman**,
 Kanata (CA)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002
 WO WO 2006/039181 A 4/2006

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

OTHER PUBLICATIONS

(*) Notice: Subject to any disclaimer, the term of this
 patent is extended or adjusted under 35
 U.S.C. 154(b) by 1293 days.

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

* cited by examiner

(21) Appl. No.: **10/946,536**

Primary Examiner—Hyung S Sough

Assistant Examiner—Syed Roni

(22) Filed: **Sep. 21, 2004**

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt,
 Milbrath & Gilchrist, P.A. Attorneys at Law

(65) **Prior Publication Data**

US 2005/0066303 A1 Mar. 24, 2005

(57) **ABSTRACT**

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep.
 22, 2003.

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

(51) **Int. Cl.**
G06F 9/22 (2006.01)

(52) **U.S. Cl.** **719/310; 719/319**

(58) **Field of Classification Search** 719/310,
 719/319

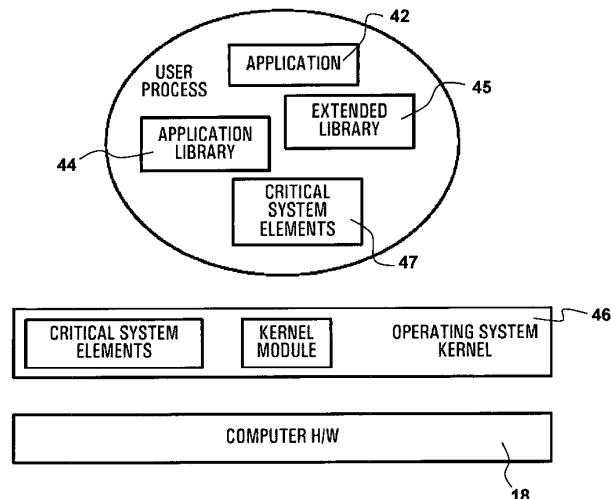
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,481,706 A * 1/1996 Peek 710/200
 6,212,574 B1 * 4/2001 O'Rourke et al. 719/321
 6,260,075 B1 * 7/2001 Cabrero et al. 719/310

18 Claims, 7 Drawing Sheets



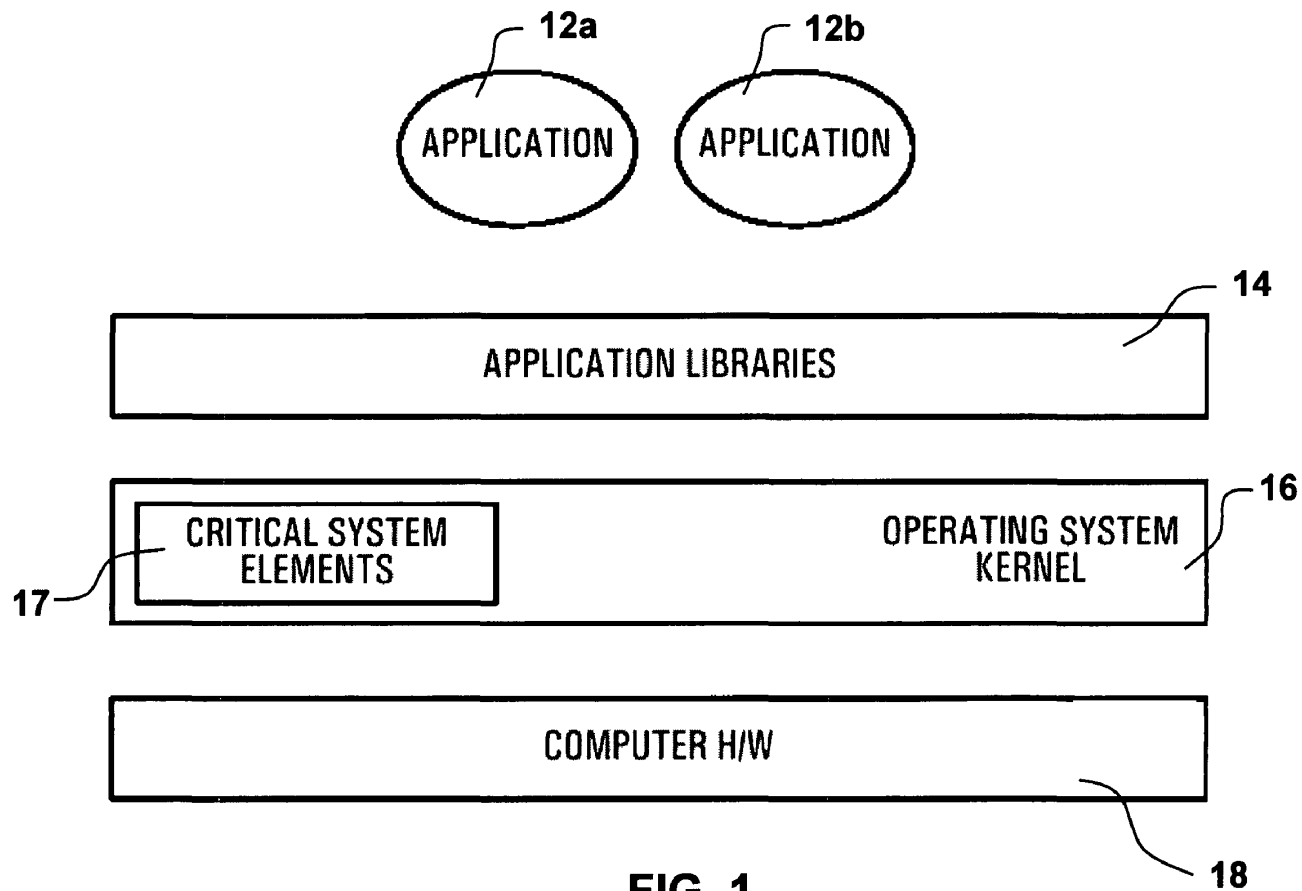


FIG. 1
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 1 of 7

US 7,784,058 B2

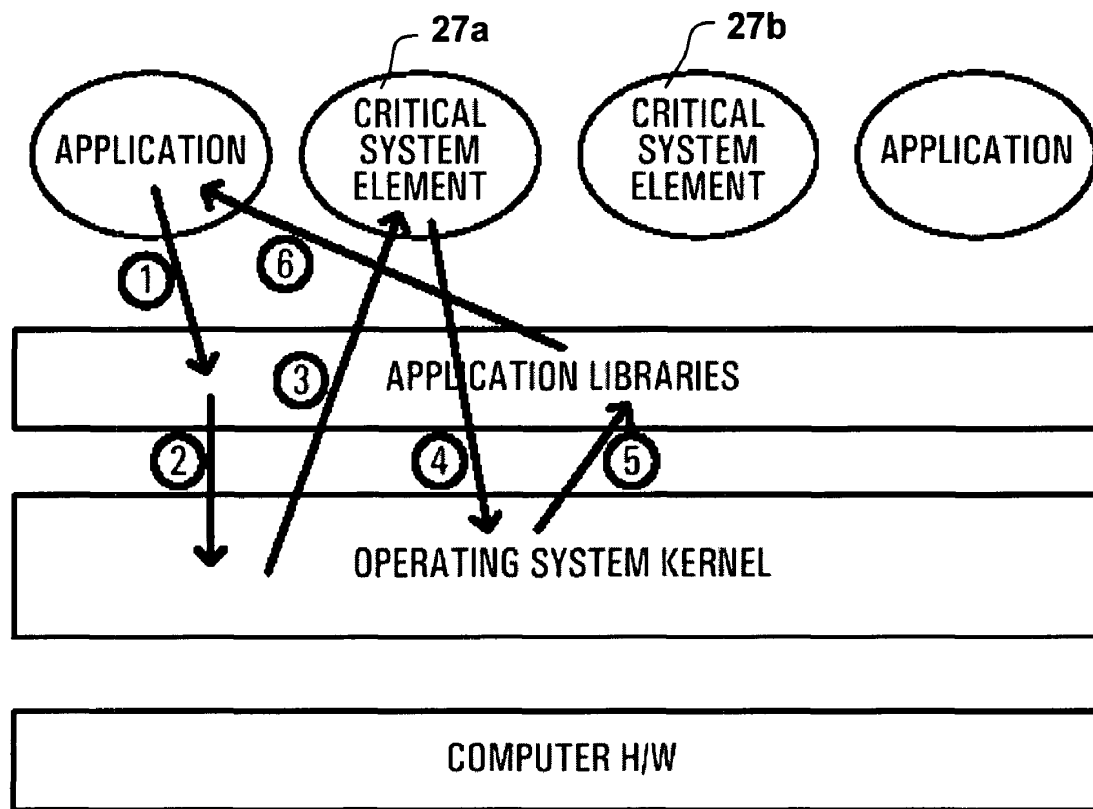


FIG. 2a
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 2 of 7

US 7,784,058 B2

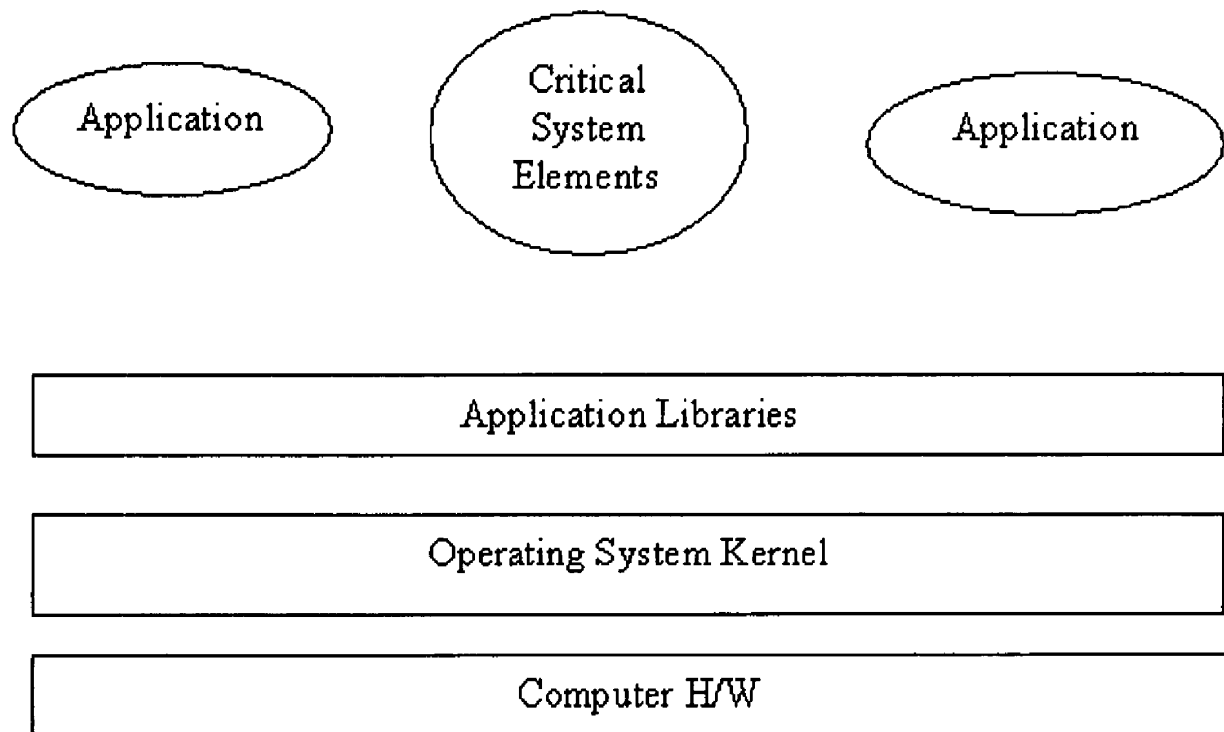


FIG. 2b
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 3 of 7

US 7,784,058 B2

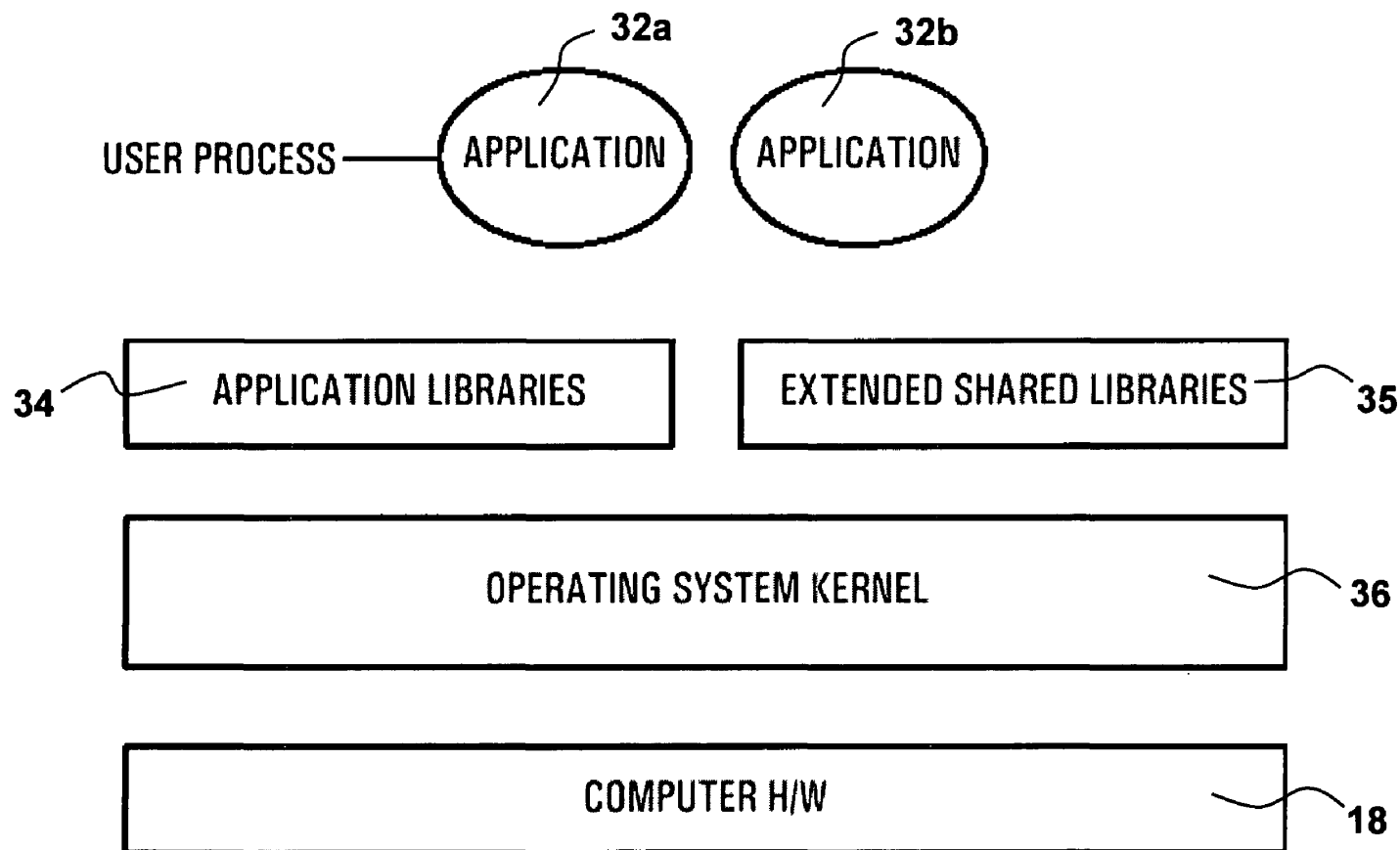


FIG. 3

U.S. Patent

Aug. 24, 2010

Sheet 4 of 7

US 7,784,058 B2

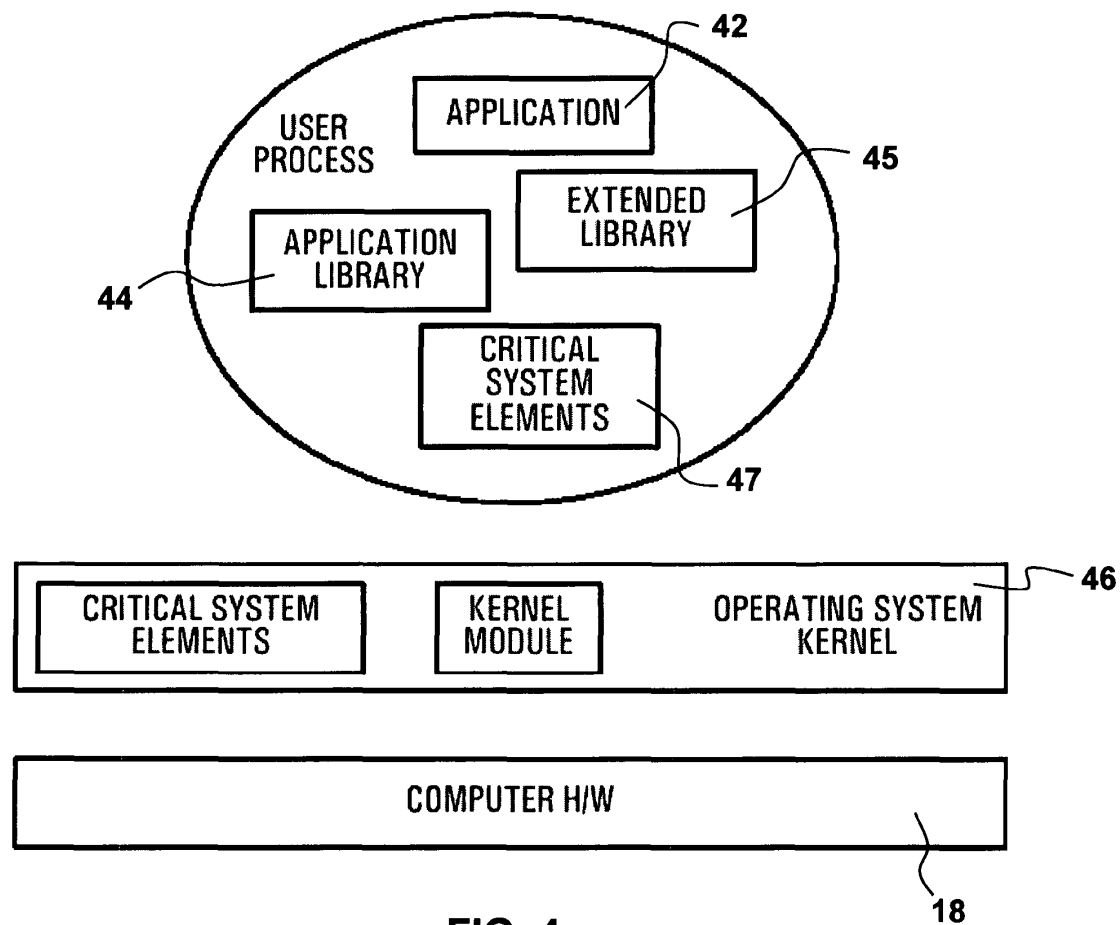


FIG. 4

U.S. Patent

Aug. 24, 2010

Sheet 5 of 7

US 7,784,058 B2

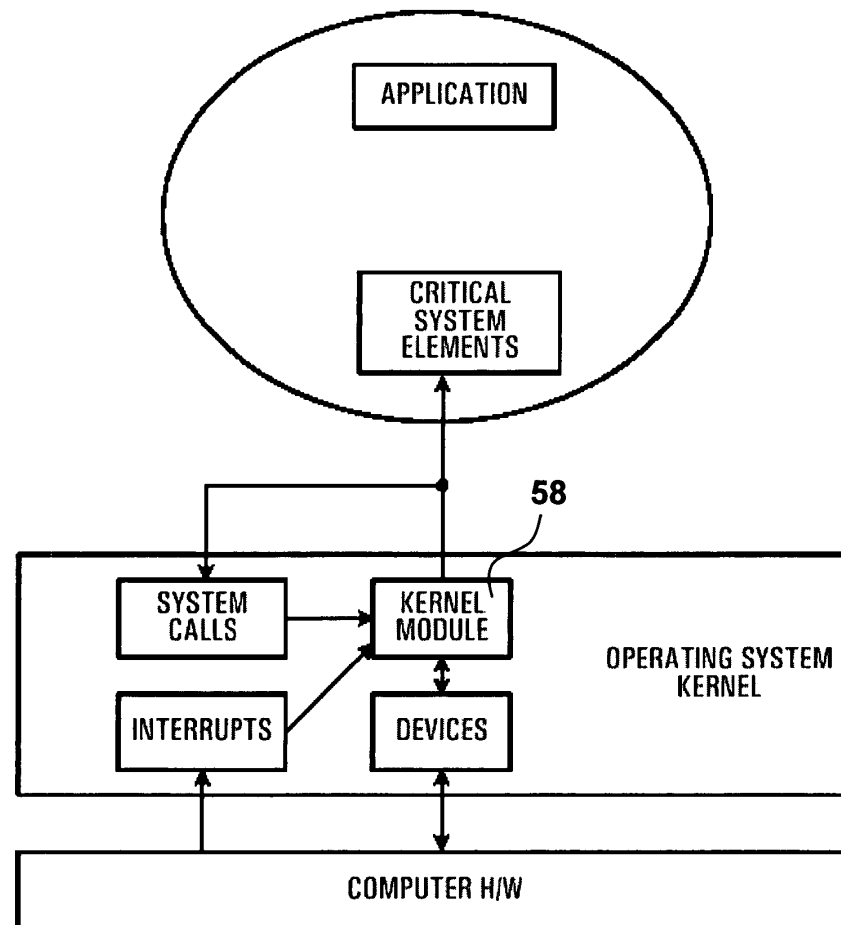


FIG. 5

U.S. Patent

Aug. 24, 2010

Sheet 7 of 7

US 7,784,058 B2

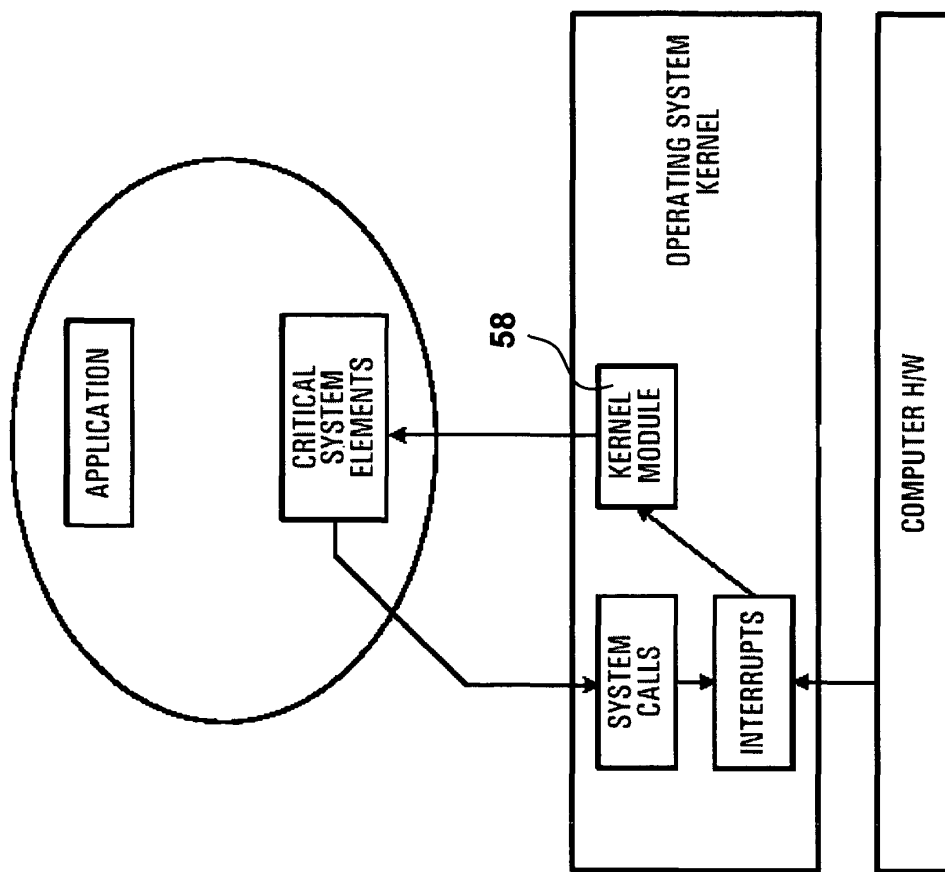


FIG. 6

US 7,784,058 B2

1

COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled "User Mode Critical System Element as Shared Libs", which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

2

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.
2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

US 7,784,058 B2

3

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

4

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. 1 is an architectural view of the traditional monolithic prior art operating system;

US 7,784,058 B2

5

FIG. 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. 3 is an architectural view of an embodiment of the invention;

FIG. 4 is a functional view showing how critical system elements exist in the same context as an application;

FIG. 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. 6 shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TCP/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP

6

stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality

2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided

2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b

US 7,784,058 B2

7

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12a**, **12b** makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12a**, **12b** in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2a** shows a system architecture where critical system elements **27a**, **27b** execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2a** is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2a** as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2b** is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2a** and FIG. **2b** is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2b** is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the

8

case with a multi-server architecture depicted in FIG. **2a**. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32a**, **32b**, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45, 47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

US 7,784,058 B2

9

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. 4. FIG. 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

10

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:

- a) a processor;
- b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,
- c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

US 7,784,058 B2

11

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

6. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

7. A computing system according to claim 6 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

8. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

12

9. A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

10. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

11. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

12. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

13. A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

14. A computing system according to claim 13 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

15. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

16. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

17. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.

* * * * *

EXHIBIT L

**UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

VIRTAMOVE, CORP.,

Plaintiff,

v.

INTERNATIONAL BUSINESS
MACHINES CORP.,

Defendant.

Case No. 2:24-cv-00064-JRG-RSP

**PLAINTIFF VIRTAMOVE, CORP.’S CORRECTED PRELIMINARY DISCLOSURE
OF ASSERTED CLAIMS AND INFRINGEMENT CONTENTIONS**

I. Patent Rule 3-1: Disclosure of Asserted Claims and Infringement Contentions

Pursuant to Patent Rule 3-1, Plaintiff VirtaMove, Corp. submits the following Preliminary Disclosure of Asserted Claims and Infringement Contentions. This disclosure is based on the information available to VirtaMove as of the date of this disclosure, and VirtaMove reserves the right to amend this disclosure to the full extent permitted, consistent with the Court’s Rules and Orders.

A. Patent Rule 3-1(a): Asserted Claims

VirtaMove asserts that Defendant International Business Machines Corp. (“Defendant” or “IBM”) infringes the following claims (collectively, “Asserted Claims”):

- (1) U.S. Patent No. 7,519,814 (“the ’814 patent”), claims 1, 2, 6, 9, and 10; and
- (2) U.S. Patent No. 7,784,058 (“the ’058 patent”), claims 1–4 and 18.

This Corrected Preliminary Disclosure of Asserted Claims and Infringement Contentions correctly reflects, consistent with the Complaint (IBM Dkt. 1) and the Amended Complaints (Consolidated Dkts. 37, 47), that the only independent claims that are asserted in this case are

independent claim 1 of the '814 patent and independent claim 1 of the '058 patent. Independent Claim 31 of the '814 patent is not, was not, and will not be asserted in this case. Otherwise, this Corrected Preliminary Disclosure of Asserted Claims and Infringement Contentions is identical to the previously served Preliminary Disclosure of Asserted Claims and Infringement Contentions.

B. Patent Rule 3-1(b): Accused Instrumentalities of which VirtaMove is aware

VirtaMove asserts that the Asserted Claims are infringed by the various instrumentalities used, made, sold, offered for sale, or imported into the United States by Defendant, including certain (a) IBM products and services using secure containerized applications, including without limitation IBM's Cloud Kubernetes Service (IKS), IBM Cloud Private (ICP), and IBM Hybrid Cloud mesh, and all versions and variations thereof since the issuance of the '814 patent; and (b) IBM products and services using user mode critical system elements as shared libraries, including without limitation IBM Cloud Kubernetes Service (IKS), IBM Cloud Private (ICP), and IBM Hybrid Cloud mesh, and all versions and variations thereof since the issuance of the '058 patent ("Accused Instrumentalities"). Defendant's Accused Instrumentalities of which VirtaMove is presently aware are described in more detail in the accompanying preliminary infringement contention charts.

VirtaMove reserves the right to accuse additional products from Defendant to the extent VirtaMove becomes aware of additional products during the discovery process. Unless otherwise stated, VirtaMove's assertions of infringement apply to all variations, versions, and applications of each of the Accused Instrumentalities, on information and belief, that different variations, versions, and applications of each of the Accused Instrumentalities are substantially the same for purposes of infringement of the Asserted Claims.

C. Patent Rule 3-1(c): Claim Charts

VirtaMove's analysis of Defendant's products is based upon limited information that is publicly available, and based on VirtaMove's own investigation prior to any discovery in these actions. Specifically, VirtaMove's analysis is based on certain limited resources that evidence certain products made, sold, used, or imported into the United States by Defendant.

VirtaMove reserves the right to amend or supplement these disclosures for any of the following reasons:

- (1) Defendant and/or third parties provide evidence relating to the Accused Instrumentalities;
- (2) VirtaMove's position on infringement of specific claims may depend on the claim constructions adopted by the Court, which has not yet occurred; and
- (3) VirtaMove's investigation and analysis of Defendant's Accused Instrumentalities is based upon public information and VirtaMove's own investigations. VirtaMove reserves the right to amend these contentions based upon discovery of non-public information that VirtaMove anticipates receiving during discovery.

Attached, and incorporated herein in their entirety, are charts identifying where each element of the Asserted Claims are found in the Accused Instrumentalities.

Unless otherwise indicated, the information provided that corresponds to each claim element is considered to indicate that each claim element is found within each of the different variations, versions, and applications of each of the respective Accused Instrumentalities described above.

D. Patent Rule 3-1(d): Literal Infringement / Doctrine of Equivalents

With respect to the patents at issue, each element of each Asserted Claim is considered to be literally present. VirtaMove also contends that each Asserted Claim is infringed or has been infringed under the doctrine of equivalents in Defendant's Accused Instrumentalities. VirtaMove

also contends that Defendant both directly and indirectly infringes the Asserted Claims. For example, the Accused Instrumentalities are provided by the Defendant to customers, who are actively encouraged and instructed (for example, through Defendant's online instructions on its website and instructions, manual, or user guides that are provided with the Accused Instrumentalities) by Defendant to use the Accused Instrumentalities in ways that directly infringe the Asserted Claims. Defendant therefore specifically intends for and induces its customers to infringe the Asserted Claims under Section 271(b) through the customers' normal and customary use of the Accused Instrumentalities. In addition, Defendant is contributorily infringing the Asserted Claims under Section 271(c) and/or Section 271(f) by selling, offering for sale, or importing the Accused Instrumentalities into the United States, which constitute a material part of the inventions claimed in the Asserted Claims, are especially made or adapted to infringe the Asserted Claims, and are otherwise not staple articles or commodities of commerce suitable for non-infringing use.

E. Patent Rule 3-1(e): Priority Dates

The Asserted Claims of the '814 patent are entitled to a priority date at least as early as September 15, 2003, the filing date of provisional application No. 60/502,619.

The Asserted Claims of the '058 patent are entitled to a priority date at least as early as September 22, 2003, the filing date of provisional application No. 60/504,213.

A diligent search continues for additional responsive information and VirtaMove reserves the right to supplement this response.

F. Patent Rule 3-1(f): Identification of Instrumentalities Practicing the Claimed Invention

At this time, VirtaMove does not identify any of its instrumentalities as practicing the Asserted Claims. A diligent search continues for additional responsive information and VirtaMove reserves the right to supplement this response.

II. Patent Rule 3-2: Document Production Accompanying Disclosure

Pursuant to Patent Rule 3-2, VirtaMove submitted the following Document Production Accompanying Disclosure, along with an identification of the categories to which each of the documents corresponds.

F. Patent Rule 3-2(a) documents:

VirtaMove is presently unaware of any documents sufficient to evidence any discussion with, disclosure to, or other manner of providing to a third party, or sale of or offer to sell, the inventions recited in the Asserted Claims of the Asserted Patents prior to the application dates or priority dates for the Asserted Patents. A diligent search continues for such documents and VirtaMove reserves the right to supplement this response.

G. Patent Rule 3-2(b) documents:

VirtaMove identifies the following non-privileged documents as related to evidencing conception and reduction to practice of each claimed invention of the Asserted Patents: VM_HPE_0000865–VM_HPE_0000880. A diligent search continues for additional documents and VirtaMove reserves the right to supplement this response.

H. Patent Rule 3-2(c) documents:

VirtaMove identifies the following documents as being the file histories for the Asserted Patents: VM_HPE_0000001–VM_HPE_0000864.

Dated: July 1, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie
CA State Bar No. 246953
Marc A. Fenster
CA State Bar No. 181067
Neil A. Rubin
CA State Bar No. 250761
Amy E. Hayden
CA State Bar No. 287026
Jacob R. Buczko
CA State Bar No. 269408
James S. Tsuei
CA State Bar No. 285530
James A. Milkey
CA State Bar No. 281283
Christian W. Conkle
CA State Bar No. 306374
Jonathan Ma
CA State Bar No. 312773
Daniel Kolko (CA SBN 341680)
RUSS AUGUST & KABAT
12424 Wilshire Boulevard, 12th Floor
Los Angeles, CA 90025
Telephone: 310-826-7474
Email: rmirzaie@raklaw.com
Email: mfenster@raklaw.com
Email: nrubin@raklaw.com
Email: ahayden@raklaw.com
Email: jbuczko@raklaw.com
Email: jtsuei@raklaw.com
Email: jmilkey@raklaw.com
Email: cconkle@raklaw.com
Email: jma@raklaw.com
Email: dkolko@raklaw.com

Qi (Peter) Tong
4925 Greenville Ave., Suite 200
Dallas, TX 75206
Email: ptong@raklaw.com

**ATTORNEYS FOR PLAINTIFF
VIRTAMOVE, CORP.**

CERTIFICATE OF SERVICE

I certify that this document is being served upon counsel of record for Defendants
on July 1, 2024 via e-mail.

/s/ Reza Mirzaie
Reza Mirzaie

U.S. Patent No. 7,519,814 (“’814 Patent”)

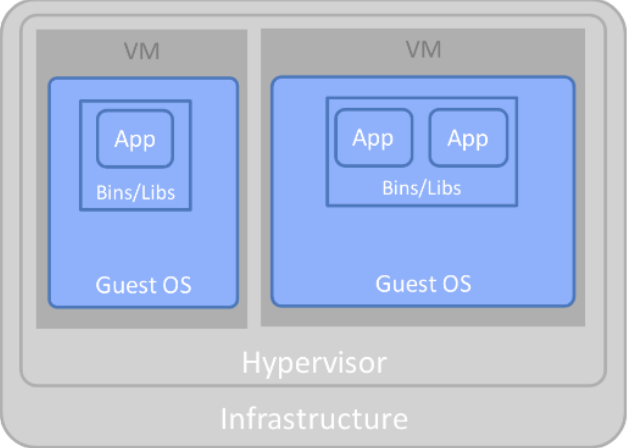
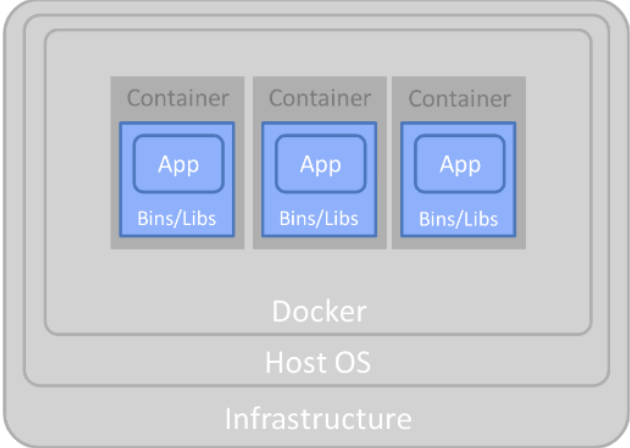
Accused Instrumentalities: IBM products and services using secure containerized applications, including without limitation IBM’s Cloud Kubernetes Service (IKS), IBM Cloud Private (ICP), and IBM Hybrid Cloud mesh, and all versions and variations thereof since the issuance of the asserted patent.

Each Accused Instrumentality infringes the claims in substantially the same way, and the evidence shown in this chart is similarly applicable to each Accused Instrumentality. Each claim limitation is literally infringed by each Accused Instrumentality. However, to the extent any claim limitation is not met literally, it is nonetheless met under the doctrine of equivalents because the differences between the claim limitation and each Accused Instrumentality would be insubstantial, and each Accused Instrumentality performs substantially the same function, in substantially the same way, to achieve the same result as the claimed invention. Notably, Defendant has not yet articulated which, if any, particular claim limitations it believes are not met by the Accused Instrumentalities.

Claim 1

Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each</p>	<p>To the extent the preamble is limiting, IBM practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p>For example, IBM Cloud Kubernetes Service runs on individual servers, each of which runs an independent operating system running either on bare metal, through an on-premises virtualized infrastructure, through one or more cloud services, or through any other supported deployment.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p>

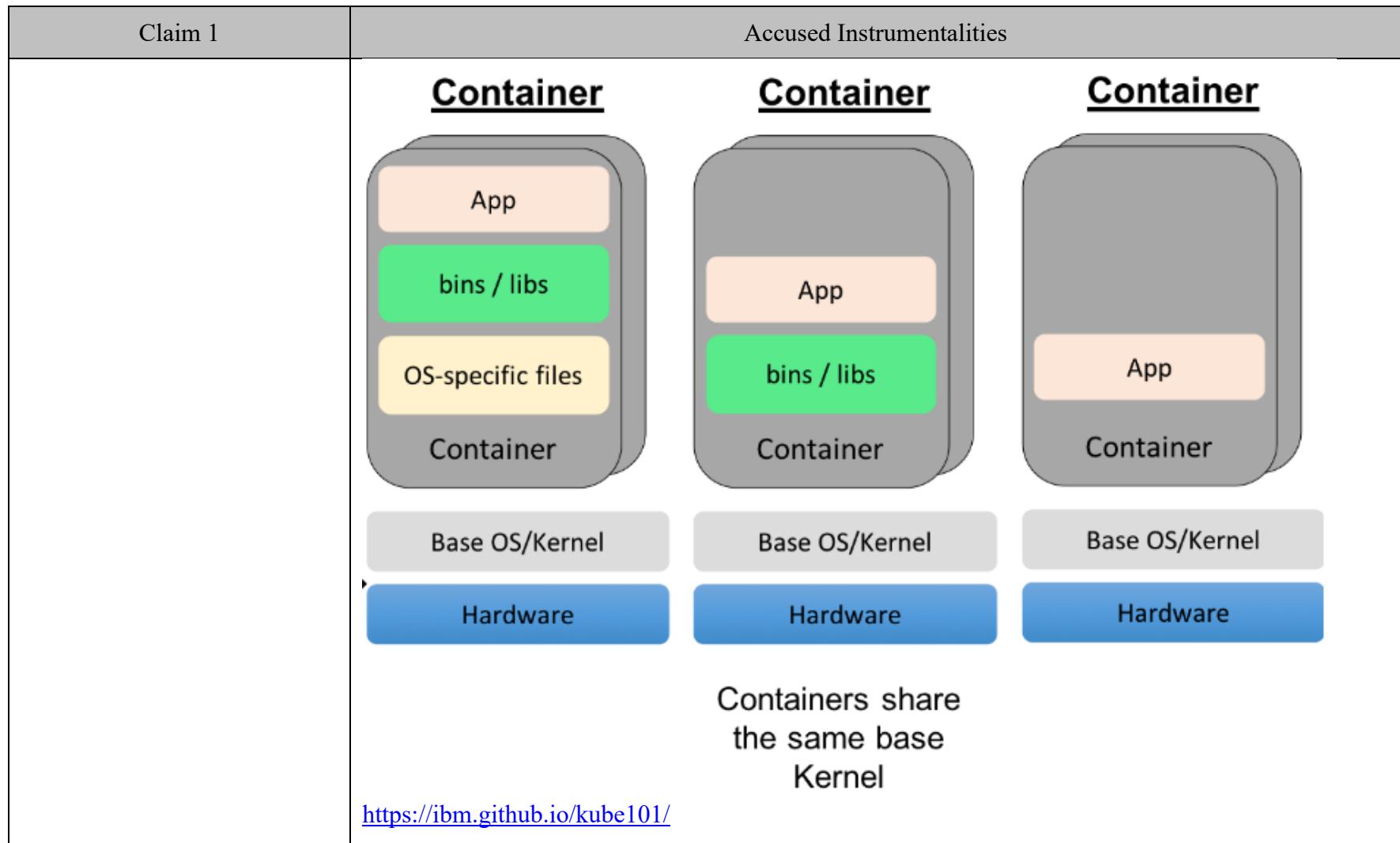
Claim 1	Accused Instrumentalities
<p>include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>IBM Cloud® Kubernetes Service provides a fully managed container service for Docker (OCI) containers, so clients can deploy containerized apps onto a pool of compute hosts and subsequently manage those containers. Containers are automatically scheduled and placed onto available compute hosts based on your requirements and availability in the cluster.</p> <p>https://www.ibm.com/products/kubernetes-service</p> <p>With IBM Cloud Kubernetes Service, you can deploy Docker containers into pods that run on your worker nodes. The worker nodes come with a set of add-on pods to help you manage your containers. Install more add-ons through Helm, a Kubernetes package manager. These add-ons can extend your apps with dashboards, logging, IBM Cloud and IBM Watson® services and more.</p> <p>https://www.ibm.com/products/kubernetes-service</p> <p>Docker is an open source platform that enables developers to build, deploy, run, update and manage <i>containers</i>—standardized, executable components that combine application source code with the operating system (OS) libraries and dependencies required to run that code in any environment.</p> <p>https://www.ibm.com/topics/docker</p>

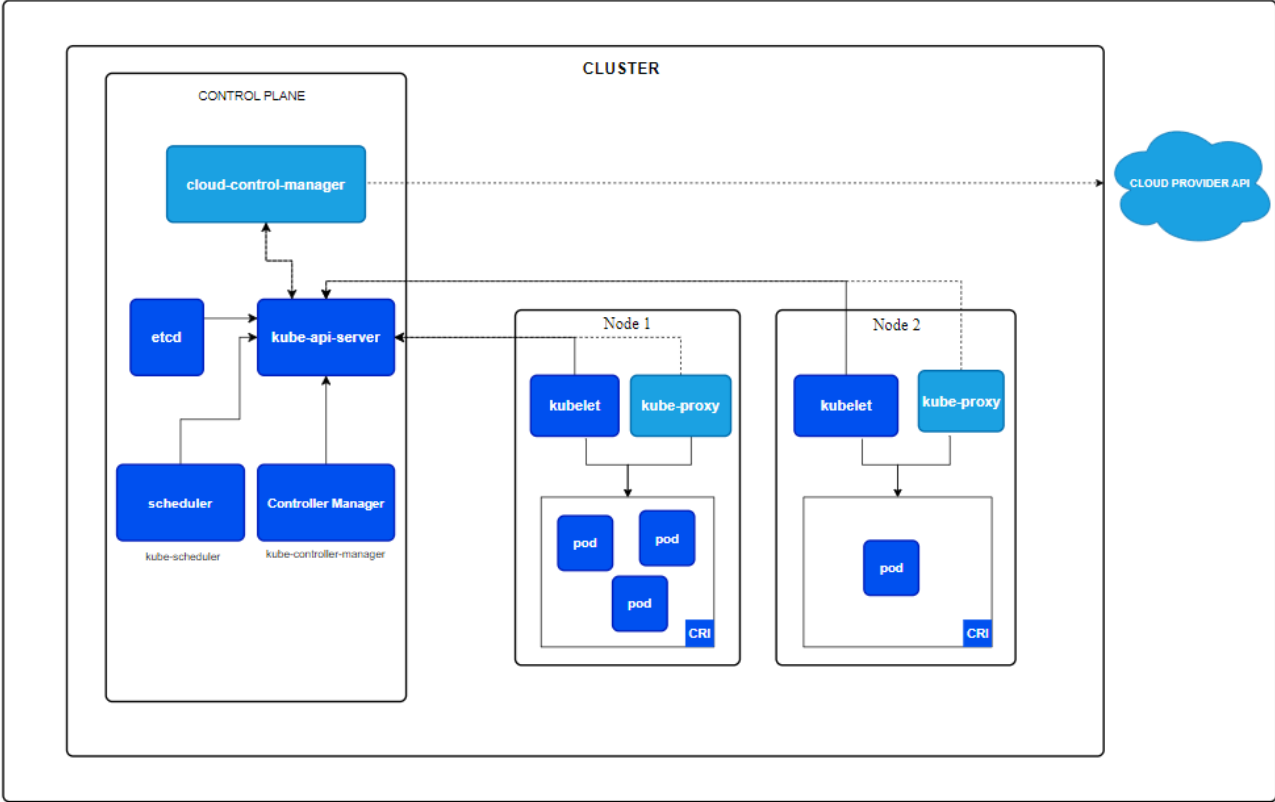
Claim 1	Accused Instrumentalities
	<div data-bbox="646 289 1915 799"> <div>Virtual Machines</div>  <div>Containers</div>  </div> <p>https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/</p>

Claim 1	Accused Instrumentalities
	<p>Containers are executable units of software in which application code is packaged along with its libraries and dependencies, in common ways so that the code can be run anywhere—whether it be on desktop, traditional IT or the cloud.</p> <p>To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (e.g. Linux namespaces and cgroups, Windows silos and job objects) can be leveraged to isolate processes and control the amount of CPU, memory and disk that those processes can access.</p> <p>Containers are small, fast and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can instead simply leverage the features and resources of the host OS.</p> <p>https://www.ibm.com/topics/containers</p> <p>With containers, you can isolate the ecosystem to run an application on any host OS (operating system). Containers can wrap code, runtimes, system tools, system libraries—everything that can be installed on a server. Containers are like virtual machines (VMs), but with a key difference in their architectural approach. Images that run on VMs have a full copy of the guest OS, including the necessary binaries and libraries. Images that run on containers share the OS kernel on the host.</p> <p>The Docker Engine builds and spins images on the containers. The engine is a lightweight container runtime that can run on almost any OS. You can run a container anywhere that a Docker Engine can be installed—on bare metal servers, clouds, and even inside a VM. You can move containers from one environment to another without recoding the application.</p> <p>Containers can help DevOps teams in three ways:</p> <ul style="list-style-type: none"> • Increase development productivity by reducing the time spent on environment setup • Eliminate issues that are caused by software dependencies • Avoid inconsistencies when applications are run in different environments <p>You can use IBM Cloud Kubernetes Service to run containers on IBM Cloud.</p> <p>https://www.ibm.com/garage/method/practices/run/tool_ibm_container/, last accessed on Nov. 17, 2023.</p>

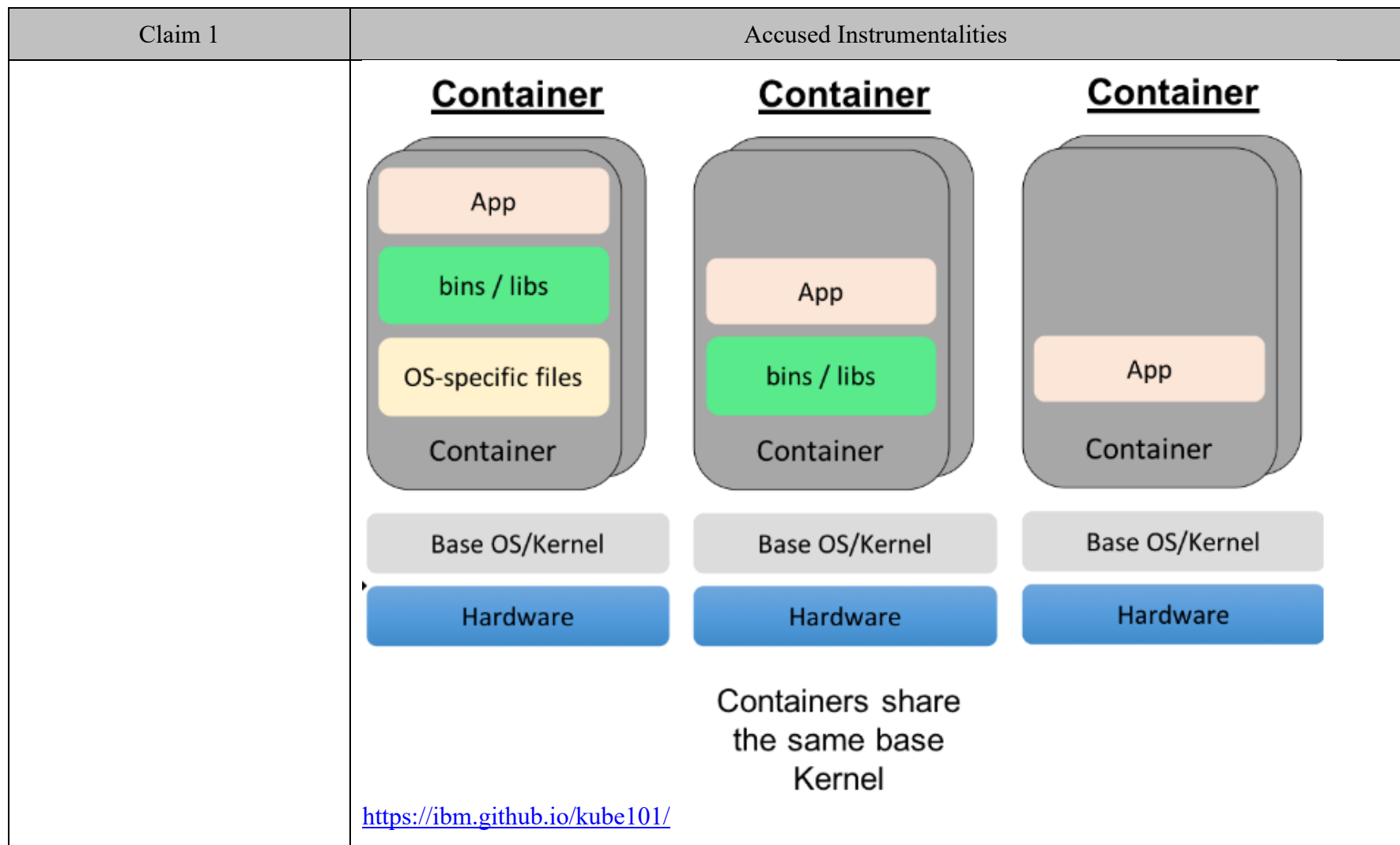
Claim 1	Accused Instrumentalities
	<p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and disk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p> <p>Today Docker containerization also works with Microsoft Windows and Apple MacOS. Developers can run Docker containers on any operating system, and most leading cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to help developers build, deploy and run applications containerized with Docker.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.</p> <p>https://www.ibm.com/topics/containerization</p>








Claim 1	Accused Instrumentalities
	 <p>The diagram illustrates the Kubernetes cluster architecture. It is divided into a 'CONTROL PLANE' and 'Node 1' and 'Node 2'. The 'CONTROL PLANE' contains components: 'cloud-control-manager', 'etcd', 'kube-api-server', 'scheduler' (labeled 'kube-scheduler'), and 'Controller Manager' (labeled 'kube-controller-manager'). The 'cloud-control-manager' is connected to 'etcd' and 'kube-api-server'. 'etcd' is connected to 'kube-api-server'. 'scheduler' is connected to 'kube-api-server'. 'Controller Manager' is connected to 'kube-api-server'. 'kube-api-server' is connected to 'cloud-control-manager'. 'Node 1' and 'Node 2' each contain 'kubelet' and 'kube-proxy'. 'kubelet' is connected to 'kube-proxy'. 'kubelet' is connected to 'pod' components. 'Node 1' has three 'pod' components, and 'Node 2' has one 'pod' component. A 'CLOUD PROVIDER API' is shown as a cloud icon connected to the 'cloud-control-manager' via a dashed line. The entire cluster is labeled 'CLUSTER'.</p> <p>Kubernetes cluster architecture https://kubernetes.io/docs/concepts/architecture/</p>
[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container	The method practiced by IBM through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.

Claim 1	Accused Instrumentalities
<p>comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>For example, IBM Cloud Kubernetes stores application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent storage available to each node running the application. The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network. Each container includes the application software as well as a Linux user space required to execute the application, for example libc/glibc and other shared libraries, configuration files, etc. necessary for the application. For example, the container includes a base OS image, provided by IBM or by a third party, such as a CentOS, RHEL, or Ubuntu base image. The container is compatible with the host kernel, for example because the container libraries are linked against the Linux kernel, and the supported host operating systems also use the Linux kernel, which has a stable binary interface.</p> <p><i>See, e.g.:</i></p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and disk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p> <p>Today Docker containerization also works with Microsoft Windows and Apple MacOS. Developers can run Docker containers on any operating system, and most leading cloud providers, including Amazon Web Services (AWS), Microsoft Azure, and IBM Cloud offer specific services to help developers build, deploy and run applications containerized with Docker.</p> <p>https://www.ibm.com/topics/docker</p>



Claim 1	Accused Instrumentalities
	<p data-bbox="667 272 982 321">Container images</p> <p data-bbox="667 349 1266 475">A <a data-bbox="667 349 835 373" href="https://kubernetes.io/docs/concepts/containers/">container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p data-bbox="636 500 1230 532"><a data-bbox="636 500 1230 532" href="https://kubernetes.io/docs/concepts/containers/">https://kubernetes.io/docs/concepts/containers/</p> <p data-bbox="661 581 1717 605">A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p data-bbox="661 646 1890 751">An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p data-bbox="636 824 1449 857"><a data-bbox="636 824 1449 857" href="https://cloud.ibm.com/docs/containers?topic=containers-images">https://cloud.ibm.com/docs/containers?topic=containers-images</p>

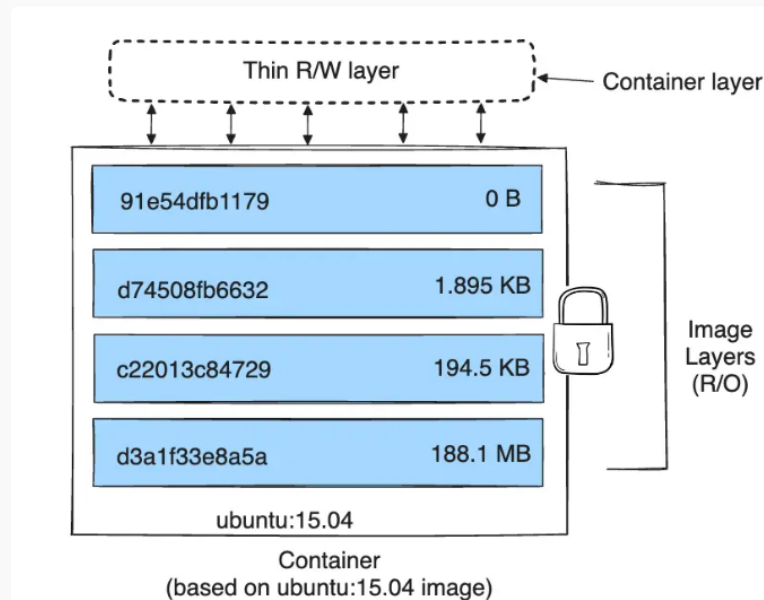
Claim 1	Accused Instrumentalities		
	Docker base image	Supported versions	Source of security notices
	Alpine	All stable versions with vendor security support.	Alpine SecDB database  .
	Debian	<p>All stable versions with vendor security support.</p> <p>CVEs on binary packages that are associated with the Debian source package <code>linux</code>, such as <code>linux-libc-dev</code>, are not reported. Most of these binary packages are kernel and kernel modules, which are not run in container images.</p>	Debian Security Bug Tracker  .
	GoogleContainerTools distroless	All stable versions with vendor security support.	GoogleContainerTools distroless  .
	Red Hat® Enterprise Linux® (RHEL)	RHEL/UBI 7, RHEL/UBI 8, and RHEL/UBI 9	Red Hat Security Data API  .
	Ubuntu	All stable versions with vendor security support.	Ubuntu CVE Tracker  .
	https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&interface=ui		

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 277 1272 342">About storage drivers</h2> <p data-bbox="646 391 1871 516">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 581 1564 646">Storage drivers versus Docker volumes</h2> <p data-bbox="646 678 1913 943">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 992 1906 1117">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1040 1535 1068" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1149 1224 1182"><a data-bbox="636 1149 1224 1182" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="655 272 1081 329">Images and layers</h2> <p data-bbox="655 367 1822 443">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="655 506 1453 824"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="655 889 1900 1198">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="634 1219 1224 1252">https://docs.docker.com/storage/storagedriver/</p>

Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles](#) and [use multi-stage builds](#) sections to learn how to optimize your Dockerfiles for efficient images.

The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.



<https://docs.docker.com/storage/storagedriver/>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 289 919 349">Volumes</h2> <p data-bbox="653 402 1906 532">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="634 557 1308 589">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="653 638 1226 690">Container environment</h2> <p data-bbox="653 727 1474 792">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="695 829 1451 992" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="634 1024 1528 1057">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 277 877 342">Images</h2> <p data-bbox="659 375 1522 526">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="659 565 1530 634">You typically create a container image of your application and push it to a registry before referring to it in a <u>Pod</u>.</p> <p data-bbox="634 662 1329 695">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="653 737 919 802">Volumes</h2> <p data-bbox="653 834 1530 1279">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <u>Pod</u> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <u>volume</u> abstraction solves both of these problems. Familiarity with <u>Pods</u> is suggested.</p> <p data-bbox="634 1307 1308 1339">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="659 282 1299 344"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="659 404 1186 454"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="659 498 1904 574"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="659 607 1908 683"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="625 708 1482 781"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

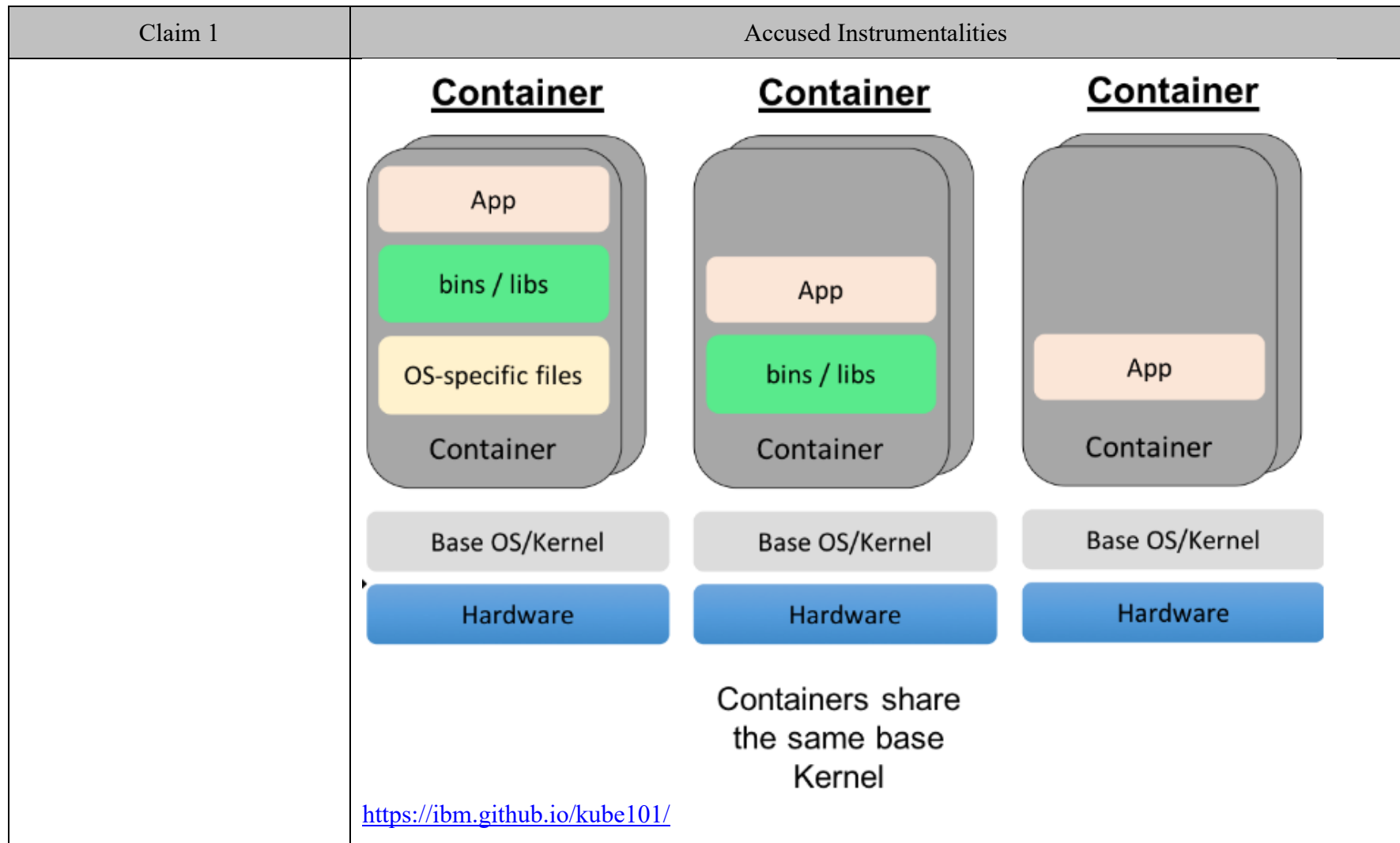
Claim 1	Accused Instrumentalities
	<p>Overview</p> <p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="661 649 1911 1039"> <pre> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p>→</p> <div style="display: flex; align-items: center; justify-content: space-around;"> <div style="border: 1px solid black; border-radius: 50%; padding: 10px; text-align: center;"> <div style="background-color: #4a7ebb; color: white; padding: 2px 5px; font-weight: bold;">Ci</div> <div style="text-align: left;">/bin/java /opt/app.jar /lib/libc</div> </div> <div>+</div> <div style="border: 1px solid black; padding: 10px; text-align: center;"> <div style="background-color: #4a7ebb; color: white; padding: 2px 5px; font-weight: bold;">Ci</div> <pre> { "manifests": { "platform": { "os": "linux", ... } } } </pre> </div> <div>+</div> <div style="border: 1px solid black; padding: 10px; text-align: center;"> <div style="background-color: #4a7ebb; color: white; padding: 2px 5px; font-weight: bold;">Ci</div> <pre> { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } } </pre> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> layer image index config </div> </div> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 272 1297 334">OCI Image Configuration</h2> <p data-bbox="653 386 1913 548">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="653 586 1661 621">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="632 651 1503 719">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>






Claim 1	Accused Instrumentalities
	<p>Layer</p> <ul style="list-style-type: none"> • Image filesystems are composed of <i>layers</i>. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p>Image JSON</p> <ul style="list-style-type: none"> • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none"> • rootfs object, REQUIRED <p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p> <ul style="list-style-type: none"> ◦ type string, REQUIRED <p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p> <ul style="list-style-type: none"> ◦ diff_ids array of strings, REQUIRED <p>An array of layer content hashes (<code>DiffIDs</code>), in order from first to last.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>
[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,	<p>In the method practiced by IBM through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.</p> <p>The system files in the container are compatible with the host kernel, for example because they are linked against the Linux kernel and the supported host operating systems also use the Linux kernel, which has a stable binary interface.</p> <p><i>See, e.g.:</i></p>

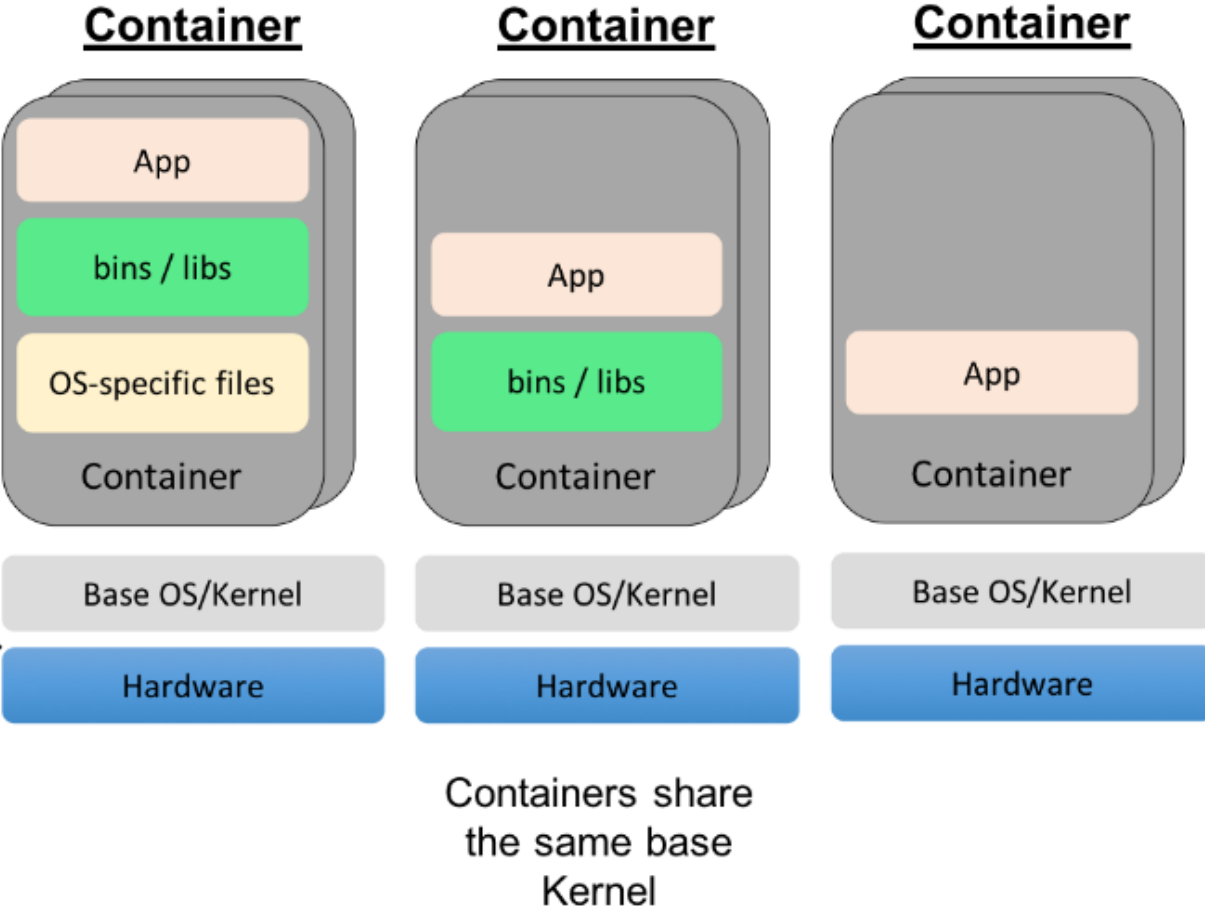
Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>Containers encapsulate an application as a single executable package of software that bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run. Containerized applications are “isolated” in that they do not bundle in a copy of the operating system. Instead, an open source runtime engine (such as the Docker runtime engine) is installed on the host’s operating system and becomes the conduit for containers to share an operating system with other containers on the same computing system.</p> <p>https://www.ibm.com/topics/containerization</p>



Claim 1	Accused Instrumentalities
	<p>A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p>An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-images</p>

Claim 1	Accused Instrumentalities		
	Docker base image	Supported versions	Source of security notices
	Alpine	All stable versions with vendor security support.	Alpine SecDB database  .
	Debian	All stable versions with vendor security support. CVEs on binary packages that are associated with the Debian source package <code>linux</code> , such as <code>linux-libc-dev</code> , are not reported. Most of these binary packages are kernel and kernel modules, which are not run in container images.	Debian Security Bug Tracker  .
	GoogleContainerTools distroless	All stable versions with vendor security support.	GoogleContainerTools distroless  .
	Red Hat® Enterprise Linux® (RHEL)	RHEL/UBI 7, RHEL/UBI 8, and RHEL/UBI 9	Red Hat Security Data API  .
	Ubuntu	All stable versions with vendor security support.	Ubuntu CVE Tracker  .
	https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&interface=ui		
[1c] the containers of application software excluding a kernel,	In the method practiced by IBM through the Accused Instrumentalities, the containers of application software exclude a kernel. <i>See, e.g.:</i>		

Claim 1	Accused Instrumentalities
	<p>Containers are often referred to as “lightweight,” meaning they share the machine’s operating system kernel and do not require the overhead of associating an operating system within each application. Containers are inherently smaller in capacity than a VM and require less start-up time, allowing far more containers to run on the same compute capacity as a single VM. This drives higher server efficiencies and, in turn, reduces server and licensing costs.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<div style="text-align: center;">  <p>Containers share the same base Kernel</p> <p>https://ibm.github.io/kube101/</p> </div>
[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the	In the method practiced by IBM through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.

Claim 1	Accused Instrumentalities
<p>associated local system files that remain resident on the server,</p>	<p>For example, each container will utilize its own local system files, including libraries such as libc/glibc and configuration files, not the corresponding libraries and configuration files of the host OS.</p> <p><i>See, e.g.:</i></p> <p>Rather than spinning up an entire virtual machine, containerization packages together everything needed to run a single application or microservice (along with runtime libraries they need to run). The container includes all the code, its dependencies and even the operating system itself. This enables applications to run almost anywhere — a desktop computer, a traditional IT infrastructure or the cloud.</p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p>
<p>[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p>For example, in some cases the host OS and container will use one or more identical system files, for example when both the host and the container incorporate the same Linux distribution version, or when both host and container use the same version of libc. In other cases modified copies are used instead, for example when different versions of the same library, or configuration files with different parameters, are used by the host and container.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than virtual machines (VMs), containers have become the de facto compute units of modern cloud-native applications.</p> <p>Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<p>With containers, you can isolate the ecosystem to run an application on any host OS (operating system). Containers can wrap code, runtimes, system tools, system libraries—everything that can be installed on a server. Containers are like virtual machines (VMs), but with a key difference in their architectural approach. Images that run on VMs have a full copy of the guest OS, including the necessary binaries and libraries. Images that run on containers share the OS kernel on the host.</p> <p>The Docker Engine builds and spins images on the containers. The engine is a lightweight container runtime that can run on almost any OS. You can run a container anywhere that a Docker Engine can be installed—on bare metal servers, clouds, and even inside a VM. You can move containers from one environment to another without recoding the application.</p> <p>Containers can help DevOps teams in three ways:</p> <ul style="list-style-type: none"> • Increase development productivity by reducing the time spent on environment setup • Eliminate issues that are caused by software dependencies • Avoid inconsistencies when applications are run in different environments <p>You can use IBM Cloud Kubernetes Service to run containers on IBM Cloud.</p> <p>https://www.ibm.com/garage/method/practices/run/tool_ibm_container/, last accessed on Nov. 17, 2023.</p>
<p>[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p>For example, each container has an isolated runtime environment that cannot be accessed by other containers, for example including a per-container writeable layer or other ephemeral per-container storage. For another example, when the plurality of secure containers each corresponds to a different container image, each container cannot access another container's image and therefore application software.</p> <p><i>See, e.g.:</i></p> <p>Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple virtual machines (VMs) to share the CPU, memory and other resources of a single hardware server.</p> <p>https://www.ibm.com/topics/docker</p>

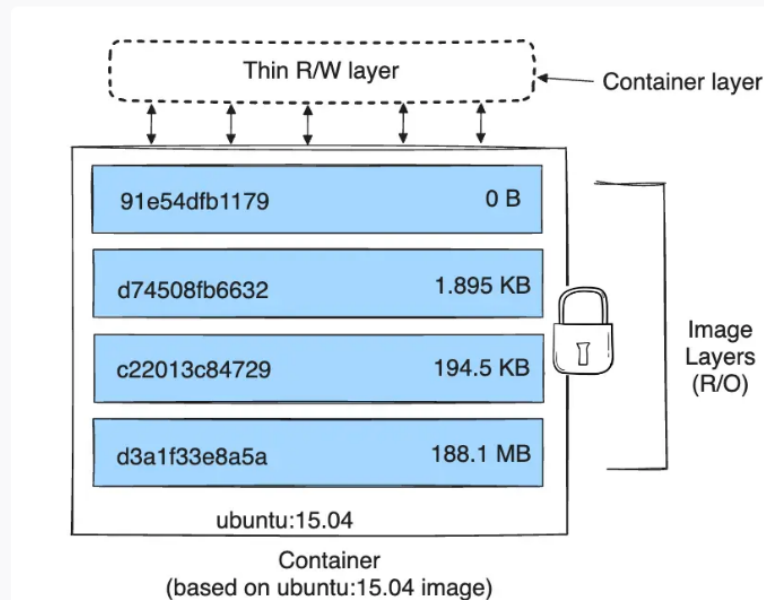
Claim 1	Accused Instrumentalities
	<p>Fault isolation: Each containerized application is isolated and operates independently of others. The failure of one container does not affect the continued operation of any other containers. Development teams can identify and correct any technical issues within one container without any downtime in other containers. Also, the container engine can leverage any OS security isolation techniques—such as SELinux access control—to isolate faults within containers.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 277 1272 342">About storage drivers</h2> <p data-bbox="646 391 1871 516">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 581 1564 646">Storage drivers versus Docker volumes</h2> <p data-bbox="646 678 1913 943">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 992 1906 1117">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1040 1535 1068" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1149 1224 1182"><a data-bbox="636 1149 1224 1182" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 272 1081 329">Images and layers</h2> <p data-bbox="657 367 1822 443">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 513 1451 824"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="657 889 1900 1198">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="634 1219 1224 1252">https://docs.docker.com/storage/storagedriver/</p>

Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles](#) and [use multi-stage builds](#) sections to learn how to optimize your Dockerfiles for efficient images.

The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.



<https://docs.docker.com/storage/storagedriver/>

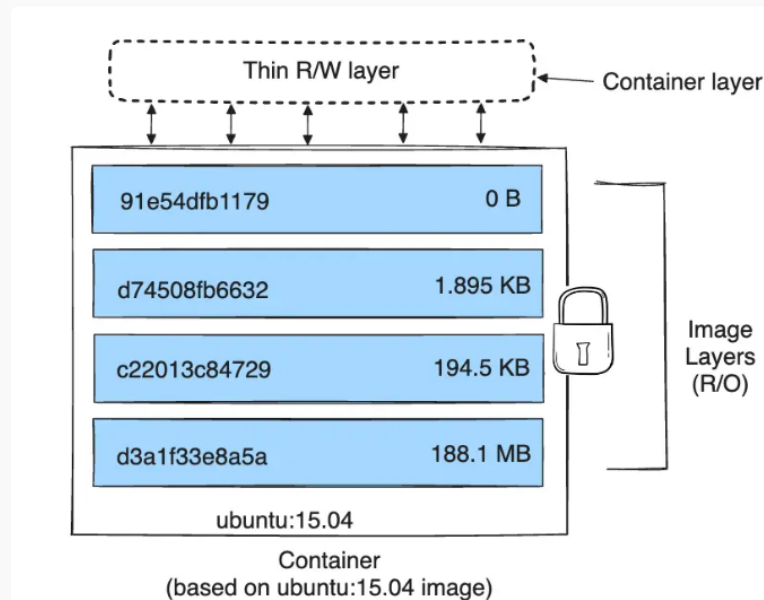
Claim 1	Accused Instrumentalities
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by IBM through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p>For example, the container's root file system comprises the image layer(s), an ephemeral writeable layer (e.g., in Docker terminology the container layer), and optionally one or more volumes. This root file system is distinct and isolated from the host operating system's root file system.</p> <p><i>See, e.g.:</i></p> <p>Limit the number of privileged containers. Containers run as a separate Linux process on the compute host that is isolated from other processes. Although users have root access inside the container, the permissions of this user are limited outside the container to protect other Linux processes, the host file system, and host devices. Some apps require access to the host file system or advanced permissions to run properly. You can run containers in privileged mode to allow the container the same access as the processes running on the compute host. Keep in mind that privileged containers can cause huge damage to the cluster and the underlying compute host if they become compromised. Try to limit the number of containers that run in privileged mode and consider changing the configuration for your app so that the app can run without advanced permissions.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-security</p> <p>Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple <i>virtual machines</i> (VMs) to share the CPU, memory and other resources of a single hardware server.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 277 1272 342">About storage drivers</h2> <p data-bbox="646 391 1871 516">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 581 1564 646">Storage drivers versus Docker volumes</h2> <p data-bbox="646 678 1913 943">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 992 1906 1117">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1040 1535 1068" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1149 1226 1182"><a data-bbox="636 1149 1226 1182" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 272 1083 329">Images and layers</h2> <p data-bbox="657 367 1822 443">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 513 1453 824"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="657 889 1900 1198">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="634 1219 1226 1252">https://docs.docker.com/storage/storagedriver/</p>

Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles](#) and [use multi-stage builds](#) sections to learn how to optimize your Dockerfiles for efficient images.

The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.



<https://docs.docker.com/storage/storagedriver/>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 289 919 349">Volumes</h2> <p data-bbox="653 402 1906 532">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="634 557 1308 589">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="653 641 1224 690">Container environment</h2> <p data-bbox="653 727 1474 792">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="695 829 1451 992" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="634 1024 1528 1057">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 280 877 342">Images</h2> <p data-bbox="659 375 1522 526">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="659 565 1530 634">You typically create a container image of your application and push it to a registry before referring to it in a <u>Pod</u>.</p> <p data-bbox="634 662 1329 695">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="653 740 919 795">Volumes</h2> <p data-bbox="653 837 1530 1279">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <u>Pod</u> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <u>volume</u> abstraction solves both of these problems. Familiarity with <u>Pods</u> is suggested.</p> <p data-bbox="634 1307 1308 1339">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="659 282 1299 344"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="659 404 1186 453"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="659 496 1904 574"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="659 607 1908 683"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="625 708 1482 781"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 1	Accused Instrumentalities
	<p>Overview</p> <p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="661 649 1911 1039"> <pre> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p>→</p> <div style="display: flex; align-items: center; justify-content: space-around;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div> </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> <div style="text-align: center;"> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> <div style="border: 1px solid black; border-radius: 50%; width: 40px; height: 40px; margin: 0 auto; display: flex; align-items: center; justify-content: center;"> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: white; margin-right: 5px;"></div> <div style="width: 10px; height: 10px; background-color: blue; margin-right: 5px;"></div> </div> </div> </div>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 272 1297 334">OCI Image Configuration</h2> <p data-bbox="653 386 1913 548">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="653 586 1661 618">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="632 651 1503 724">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="661 280 751 318">Layer</p> <ul data-bbox="688 354 1915 699" style="list-style-type: none"> • Image filesystems are composed of <i>layers</i>. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="661 751 856 789">Image JSON</p> <ul data-bbox="688 824 1915 1170" style="list-style-type: none"> • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="634 1198 1503 1268">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none"> • rootfs object, REQUIRED <p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p> <ul style="list-style-type: none"> ◦ type string, REQUIRED <p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p> <ul style="list-style-type: none"> ◦ diff_ids array of strings, REQUIRED <p>An array of layer content hashes (<code>DiffIDs</code>), in order from first to last.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 2

Claim 2	Accused Instrumentalities
2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.	<p>IBM practices, through the Accused Instrumentalities, a method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.</p> <p>For example, a container image has an associated image configuration comprising information for starting the one or more applications. This can be an Open Containers Initiative image configuration.</p> <p><i>See, e.g.:</i></p>

Claim 2	Accused Instrumentalities
	<div data-bbox="625 256 1251 315"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="625 378 1136 423"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="625 470 1848 545"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="625 579 1854 654"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="585 682 1432 751"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 2	Accused Instrumentalities
	<p data-bbox="604 250 785 289">Overview</p> <p data-bbox="604 345 1848 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="621 630 1864 1008"> <p>The diagram illustrates the components of an OCI image. On the left, a code block for a 'HelloWorld' class is shown. An arrow points from this code to a cylinder labeled 'layer' containing the paths '/bin/java', '/opt/app.jar', and '/lib/libc'. To the right of the layer is a plus sign, followed by a document icon labeled 'image index' containing a JSON snippet for 'manifests'. Another plus sign follows, leading to a document icon labeled 'config' containing a JSON snippet for 'config'. Each of these three components (layer, image index, and config) has a small square icon with the letters 'ci' in the top-left corner.</p> <pre> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <pre> { "manifests": { "platform": { "os": "linux", ... } } } </pre> <pre> { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } } </pre> <p data-bbox="588 1036 1432 1105">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p> </div>

Claim 2	Accused Instrumentalities
	<h2 data-bbox="604 245 1251 305">OCI Image Configuration</h2> <p data-bbox="604 358 1871 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="604 558 1612 591">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="588 623 1457 695">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 2	Accused Instrumentalities
	<ul style="list-style-type: none"> • config object, OPTIONAL <p>The execution parameters which SHOULD be used as a base when running a container using the image. This field can be <code>null</code>, in which case any execution parameters should be specified at creation of the container.</p> <ul style="list-style-type: none"> ◦ Env array of strings, OPTIONAL <p>Entries are in the format of <code>VARNAME=VARVALUE</code>. These values act as defaults and are merged with any specified when creating a container.</p> <ul style="list-style-type: none"> ◦ Entrypoint array of strings, OPTIONAL <p>A list of arguments to use as the command to execute when the container starts. These values act as defaults and may be replaced by an entrypoint specified when creating a container.</p> <ul style="list-style-type: none"> ◦ Cmd array of strings, OPTIONAL <p>Default arguments to the entrypoint of the container. These values act as defaults and may be replaced by any specified when creating a container. If an <code>Entrypoint</code> value is not specified, then the first entry of the <code>Cmd</code> array SHOULD be interpreted as the executable to run.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 6

Claim 6	Accused Instrumentalities
<p>6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.</p>	<p>IBM practices, through the Accused Instrumentalities, a method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.</p> <p>For example, Kubernetes containers have an associated hostname, which in the case of a single-container Pod is the unique identity of that container. For another example, Kubernetes pods have an associated hostname, which is unique. For another example, a networked Kubernetes pod has an assigned IPv4 and/or IPv6 address. For another example, a Docker container has an IP address and a hostname.</p> <p><i>See, e.g.:</i></p> <p>Container information</p> <p>The <i>hostname</i> of a Container is the name of the Pod in which the Container is running. It is available through the <code>hostname</code> command or the <code>gethostname</code> function call in libc.</p> <p>The Pod name and namespace are available as environment variables through the downward API.</p> <p>User defined environment variables from the Pod definition are also available to the Container, as are any environment variables specified statically in the container image.</p> <p>https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 6	Accused Instrumentalities
	<p data-bbox="604 245 1203 293">IP address and hostname</p> <p data-bbox="604 337 1864 461">By default, the container gets an IP address for every Docker network it attaches to. A container receives an IP address out of the IP subnet of the network. The Docker daemon performs dynamic subnetting and IP address allocation for containers. Each network also has a default subnet mask and gateway.</p> <p data-bbox="604 511 1843 683">You can connect a running container to multiple networks, either by passing the <code>--network</code> flag multiple times when creating the container, or using the <code>docker network connect</code> command for already running containers. In both cases, you can use the <code>--ip</code> or <code>--ip6</code> flags to specify the container's IP address on that particular network.</p> <p data-bbox="604 734 1850 857">In the same way, a container's hostname defaults to be the container's ID in Docker. You can override the hostname using <code>--hostname</code>. When connecting to an existing network using <code>docker network connect</code>, you can use the <code>--alias</code> flag to specify an additional network alias for the container on that network.</p> <p data-bbox="588 889 1014 919">https://docs.docker.com/network/</p>

Claim 9

Claim 9	Accused Instrumentalities
<p data-bbox="205 1081 684 1365">9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.</p>	<p data-bbox="716 1081 1854 1219">IBM practices, through the Accused Instrumentalities, a method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.</p> <p data-bbox="716 1243 1843 1349">For example, Kubernetes tracks and limits resource usage, including CPU and memory resources. For another example, Docker tracks and limits resource usage, including CPU and memory resources.</p> <p data-bbox="716 1373 831 1411"><i>See, e.g.:</i></p>

Resource Management for Pods and Containers

When you specify a Pod, you can optionally specify how much of each resource a container needs. The most common resources to specify are CPU and memory (RAM); there are others.

When you specify the resource *request* for containers in a Pod, the kube-scheduler uses this information to decide which node to place the Pod on. When you specify a resource *limit* for a container, the kubelet enforces those limits so that the running container is not allowed to use more of that resource than the limit you set. The kubelet also reserves at least the *request* amount of that system resource specifically for that container to use.

Requests and limits

If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its *request* for that resource specifies. However, a container is not allowed to use more than its resource *limit*.

For example, if you set a *memory request* of 256 MiB for a container, and that container is in a Pod scheduled to a Node with 8GiB of memory and no other Pods, then the container can try to use more RAM.

If you set a *memory limit* of 4GiB for that container, the kubelet (and container runtime) enforce the limit. The runtime prevents the container from using more than the configured resource limit. For example: when a process in the container tries to consume more than

Claim 9	Accused Instrumentalities
	<p>the allowed amount of memory, the system kernel terminates the process that attempted the allocation, with an out of memory (OOM) error.</p> <p>Limits can be implemented either reactively (the system intervenes once it sees a violation) or by enforcement (the system prevents the container from ever exceeding the limit). Different runtimes can have different ways to implement the same restrictions.</p> <p>https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</p> <p>Runtime options with Memory, CPUs, and GPUs</p> <p>By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler allows. Docker provides ways to control how much memory, or CPU a container can use, setting runtime configuration flags of the <code>docker run</code> command. This section provides details on when you should set such limits and the possible implications of setting them.</p> <p>Limit a container's access to memory</p> <p>Docker can enforce hard or soft memory limits.</p> <ul style="list-style-type: none"> • Hard limits lets the container use no more than a fixed amount of memory. • Soft limits lets the container use as much memory as it needs unless certain conditions are met, such as when the kernel detects low memory or contention on the host machine. <p>https://docs.docker.com/config/containers/resource_constraints/</p>

Claim 10

Claim 10	Accused Instrumentalities
<p>10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.</p>	<p>IBM practices, through the Accused Instrumentalities, a method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.</p> <p><i>See, e.g.:</i></p> <p>Containers are made possible by process isolation and virtualization capabilities built into the Linux kernel. These capabilities—such as <i>control groups</i> (Cgroups) for allocating resources among processes, and <i>namespaces</i> for restricting a processes access or visibility into other resources or areas of the system—enable multiple application components to share the resources of a single instance of the host operating system in much the same way that a hypervisor enables multiple virtual machines (VMs) to share the CPU, memory and other resources of a single hardware server.</p> <p>https://www.ibm.com/topics/docker</p> <p>Fault isolation: Each containerized application is isolated and operates independently of others. The failure of one container does not affect the continued operation of any other containers. Development teams can identify and correct any technical issues within one container without any downtime in other containers. Also, the container engine can leverage any OS security isolation techniques—such as SELinux access control—to isolate faults within containers.</p> <p>https://www.ibm.com/topics/containerization</p>

Claim 31

<u>Claim 31</u>	<u>Accused Instrumentalities</u>
<p><u>[31pre] A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:</u></p>	<p><u>To the extent the preamble is construed as a limitation, each Accused Instrumentality is or comprises a computing system for performing a plurality of tasks each comprising a plurality of processes.</u></p> <p><u>See claim limitations below. See also analysis and evidence for [1pre] above.</u></p>
<p><u>[31a] a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac address</u></p>	<p><u>Each Accused Instrumentality comprises a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac address.</u></p> <p><u>See analysis and evidence for [1pre], limitations [1a] and [1f], and claim 6 above.</u></p>

<u>Claim 31</u>	<u>Accused Instrumentalities</u>
<p><u>[31b] wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,</u></p>	<p><u>Each Accused Instrumentality comprises a system wherein the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel.</u></p> <p><u>See analysis and evidence for [1pre], limitations [1a], [1c], [1d], [1e], and [1f], and claim 2 above.</u></p>

~~[31c] a run-time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.~~

~~Each Accused Instrumentality comprises a run-time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.~~

~~For example, IBM Cloud Kubernetes Service includes the containerd runtime module or another container runtime. For another example, Kubernetes uses the Linux kernel's seccomp mode to monitor and control system calls made from a container.~~

~~See, e.g.:~~

Security Bulletin: IBM Cloud Kubernetes Service is affected by a containerd security vulnerability (CVE-2024-21626)

Security Bulletin

Summary

IBM Cloud Kubernetes Service is affected by a security vulnerability found in the runc component shipped with containerd where an attacker could gain unauthorized access to the host filesystem (CVE-2024-21626).

~~<https://www.ibm.com/support/pages/security-bulletin-ibm-cloud-kubernetes-service-affected-containerd-security-vulnerability-cve-2024-21626>~~

containerd Adopters

A non-exhaustive list of containerd adopters is provided below.

Docker/Moby engine - Containerd began life prior to its CNCF adoption as a lower-layer runtime manager for `runc` processes below the Docker engine. Continuing today, containerd has extremely broad production usage as a component of the [Docker engine](#) stack. Note that this includes any use of the open source [Moby engine project](#); including the Balena project listed below.

IBM Cloud Kubernetes Service (IKS) - offers containerd as the CRI runtime for v1.11 and higher versions.

IBM Cloud Private (ICP) - IBM's on-premises cloud offering has containerd as a "tech preview" CRI runtime for the Kubernetes offered within this product for the past two releases, and plans to fully migrate to containerd in a future release.

<https://github.com/moby/containerd/blob/docker/20.10/ADOPTERS.md>

Container Runtimes

Note: Dockershim has been removed from the Kubernetes project as of release 1.24. Read the [Dockershim Removal FAQ](#) for further details.

~~You need to install a container runtime into each node in the cluster so that Pods can run there. This page outlines what is involved and describes related tasks for setting up nodes.~~

Kubernetes 1.30 requires that you use a runtime that conforms with the Container Runtime Interface (CRI).

See [CRI version support](#) for more information.

~~<https://kubernetes.io/docs/setup/production-environment/container-runtimes/>~~

Claim 31Accused Instrumentalities

Restrict a Container's Syscalls with seccomp

① **FEATURE STATE:** Kubernetes v1.19 [stable]

Seccomp stands for secure computing mode and has been a feature of the Linux kernel since version 2.6.12. It can be used to sandbox the privileges of a process, restricting the calls it is able to make from userspace into the kernel. Kubernetes lets you automatically apply seccomp profiles loaded onto a node to your Pods and containers.

Identifying the privileges required for your workloads can be difficult. In this tutorial, you will go through how to load seccomp profiles into a local Kubernetes cluster, how to apply them to a Pod, and how you can begin to craft profiles that give only the necessary privileges to your container processes.

<https://kubernetes.io/docs/tutorials/security/seccomp/>

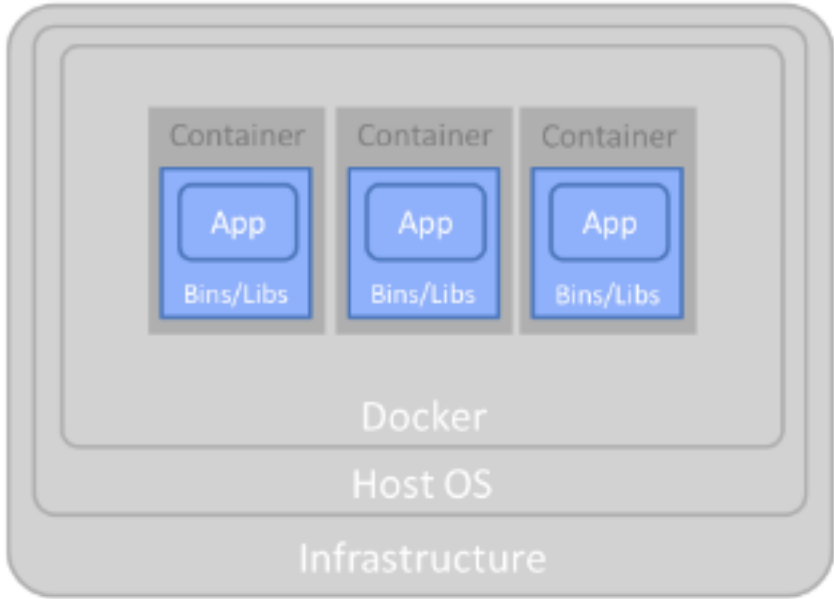
U.S. Patent No. 7,784,058 (“’058 Patent”)

Accused Instrumentalities: IBM products and services using user mode critical system elements as shared libraries, including without limitation IBM Cloud Kubernetes Service (IKS), IBM Cloud Private (ICP), and IBM Hybrid Cloud mesh,, and all versions and variations thereof since the issuance of the asserted patent.

Each Accused Instrumentality infringes the claims in substantially the same way, and the evidence shown in this chart is similarly applicable to each Accused Instrumentality. Each claim limitation is literally infringed by each Accused Instrumentality. However, to the extent any claim limitation is not met literally, it is nonetheless met under the doctrine of equivalents because the differences between the claim limitation and each Accused Instrumentality would be insubstantial, and each Accused Instrumentality performs substantially the same function, in substantially the same way, to achieve the same result as the claimed invention. Notably, Defendant has not yet articulated which, if any, particular claim limitations it believes are not met by the Accused Instrumentalities.

Claim 1






Claim 1	Accused Instrumentalities
<p>[1pre] 1. A computing system for executing a plurality of software applications comprising:</p>	<p>To the extent the preamble is limiting, each Accused Instrumentality comprises or constitutes a computing system for executing a plurality of software applications as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>IBM Cloud® Kubernetes Service provides a fully managed container service for Docker (OCI) containers, so clients can deploy containerized apps onto a pool of compute hosts and subsequently manage those containers. Containers are automatically scheduled and placed onto available compute hosts based on your requirements and availability in the cluster.</p> <p>https://www.ibm.com/products/kubernetes-service</p> <p>With IBM Cloud Kubernetes Service, you can deploy Docker containers into pods that run on your worker nodes. The worker nodes come with a set of add-on pods to help you manage your containers. Install more add-ons through Helm, a Kubernetes package manager. These add-ons can extend your apps with dashboards, logging, IBM Cloud and IBM Watson® services and more.</p> <p>https://www.ibm.com/products/kubernetes-service</p>

Claim 1	Accused Instrumentalities
	<p style="text-align: center;">Containers</p>  <p>The diagram illustrates a container architecture stack. At the base is a grey rounded rectangle labeled 'Infrastructure'. Above it is a slightly smaller grey rounded rectangle labeled 'Host OS'. Inside the Host OS is a grey rounded rectangle labeled 'Docker'. Within the Docker container, there are three separate grey rounded rectangles, each labeled 'Container'. Each 'Container' contains a blue rounded rectangle labeled 'App' and a smaller blue rounded rectangle below it labeled 'Bins/Libs'.</p> <p style="text-align: center;">https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/</p>
[1a] a) a processor;	<p>Each Accused Instrumentality comprises a processor.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<p>Containers are executable units of software in which application code is packaged along with its libraries and dependencies, in common ways so that the code can be run anywhere—whether it be on desktop, traditional IT or the cloud.</p> <p>To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (e.g. Linux namespaces and cgroups, Windows silos and job objects) can be leveraged to isolate processes and control the amount of CPU, memory and disk that those processes can access.</p> <p>Containers are small, fast and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can instead simply leverage the features and resources of the host OS.</p> <p>https://www.ibm.com/topics/containers</p> <p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p>

Claim 1	Accused Instrumentalities
<p>[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,</p>	<p>Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.</p> <p><i>See, e.g.:</i></p> <p>Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable—called a container—that runs consistently on any infrastructure. More portable and resource-efficient than <i>virtual machines (VMs)</i>, containers have become the de facto compute units of modern cloud-native applications.</p> <p>Containerization allows developers to create and deploy applications faster and more securely. With traditional methods, code is developed in a specific computing environment which, when transferred to a new location, often results in bugs and errors. For example, when a developer transfers code from a desktop computer to a VM or from a Linux to a Windows operating system. Containerization eliminates this problem by bundling the application code together with the related configuration files, libraries, and dependencies required for it to run. This single package of software or “container” is abstracted away from the host operating system, and hence, it stands alone and becomes portable—able to run across any platform or cloud, free of issues.</p> <p>https://www.ibm.com/topics/containerization</p> <p>Kernel mode</p> <p>Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p>https://www.techtarget.com/searchdatacenter/definition/kernel</p>

Claim 1	Accused Instrumentalities
	<p>The GNU C Library, commonly known as glibc, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system.</p> <p>https://en.wikipedia.org/wiki/Glibc</p>
<p>[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and</p>	<p>Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.</p> <p><i>See, e.g.:</i></p> <p>A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p>An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-images</p>

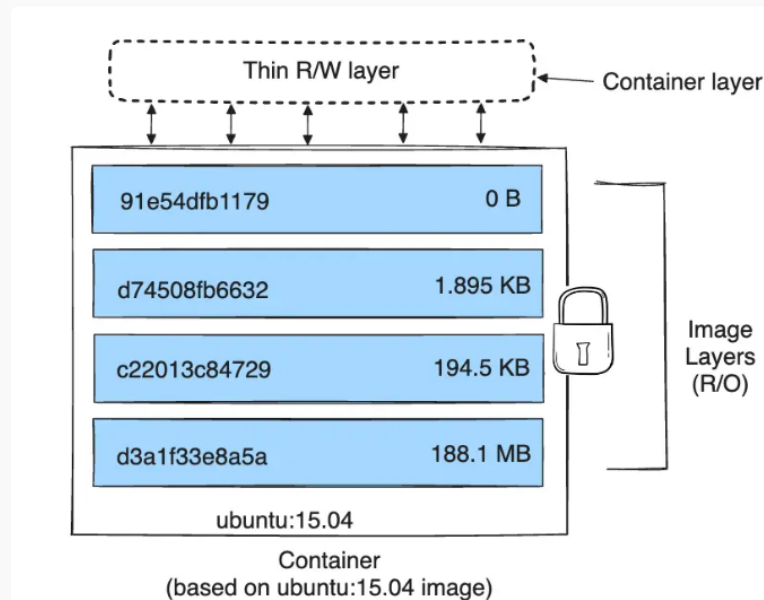
Claim 1	Accused Instrumentalities		
	Docker base image	Supported versions	Source of security notices
	Alpine	All stable versions with vendor security support.	Alpine SecDB database  .
	Debian	All stable versions with vendor security support. CVEs on binary packages that are associated with the Debian source package <code>linux</code> , such as <code>linux-libc-dev</code> , are not reported. Most of these binary packages are kernel and kernel modules, which are not run in container images.	Debian Security Bug Tracker  .
	GoogleContainerTools distroless	All stable versions with vendor security support.	GoogleContainerTools distroless  .
	Red Hat® Enterprise Linux® (RHEL)	RHEL/UBI 7, RHEL/UBI 8, and RHEL/UBI 9	Red Hat Security Data API  .
	Ubuntu	All stable versions with vendor security support.	Ubuntu CVE Tracker  .
	https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&interface=ui		

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 280 1272 349">About storage drivers</h2> <p data-bbox="646 394 1871 521">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 589 1564 646">Storage drivers versus Docker volumes</h2> <p data-bbox="646 683 1913 950">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 1000 1902 1125">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1047 1535 1073" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1154 1224 1187"><a data-bbox="636 1154 1224 1187" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 280 1081 334">Images and layers</h2> <p data-bbox="657 371 1822 448">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="674 516 1451 829"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="657 894 1900 1203">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="636 1224 1224 1256">https://docs.docker.com/storage/storagedriver/</p>

Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles](#) and [use multi-stage builds](#) sections to learn how to optimize your Dockerfiles for efficient images.

The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.

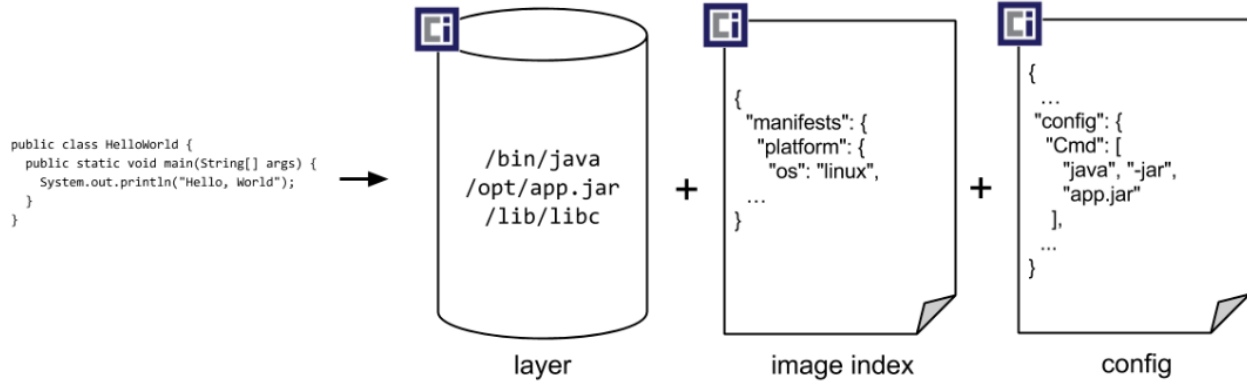





<https://docs.docker.com/storage/storagedriver/>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 289 919 354">Volumes</h2> <p data-bbox="653 407 1906 537">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="634 558 1308 591">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="653 643 1226 691">Container environment</h2> <p data-bbox="653 729 1474 797">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="695 834 1451 992" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="634 1029 1528 1062">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 285 877 345">Images</h2> <p data-bbox="659 378 1522 532">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="659 568 1528 641">You typically create a container image of your application and push it to a registry before referring to it in a <u>Pod</u>.</p> <p data-bbox="634 667 1329 703">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="653 743 919 803">Volumes</h2> <p data-bbox="653 841 1482 914">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers.</p> <p data-bbox="653 922 1430 954">One problem occurs when a container crashes or is stopped.</p> <p data-bbox="653 963 1528 1284">Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <u>Pod</u> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <u>volume</u> abstraction solves both of these problems. Familiarity with <u>Pods</u> is suggested.</p> <p data-bbox="634 1310 1308 1346">https://kubernetes.io/docs/concepts/storage/volumes/</p>

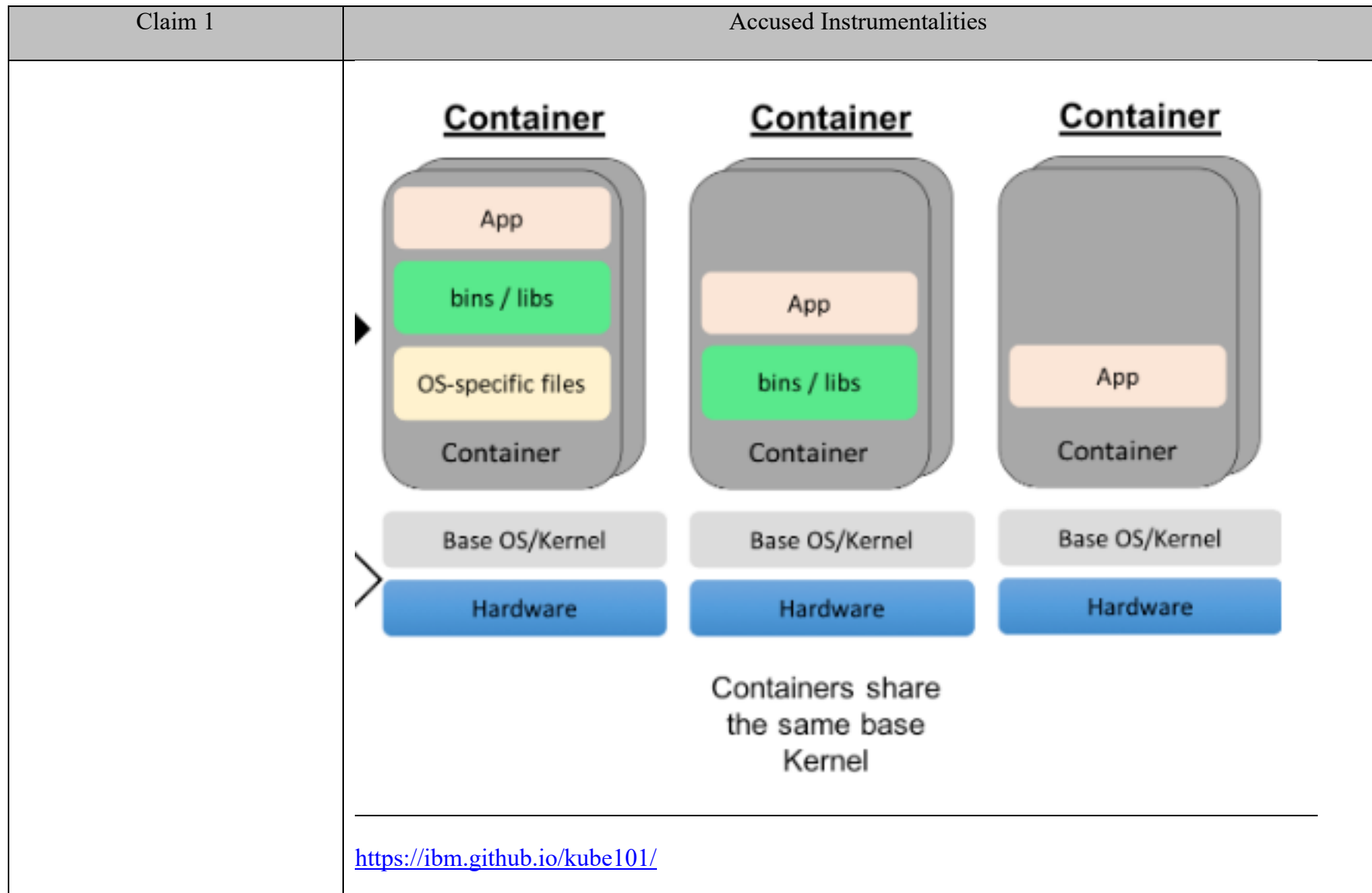
Claim 1	Accused Instrumentalities
	<div data-bbox="659 289 1299 349"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="659 410 1186 456"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="659 503 1904 578"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="659 612 1908 686"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="625 712 1482 784"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 1	Accused Instrumentalities
	<p>Overview</p> <p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p>  <pre> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p>→</p> <div style="display: flex; align-items: center; justify-content: space-around;"> <div style="text-align: center;">  <div style="border: 1px solid black; border-radius: 50%; padding: 10px; width: 100px; height: 100px; margin: 0 auto; display: flex; flex-direction: column; justify-content: center; align-items: center;"> <div style="font-size: 0.8em;">/bin/java</div> <div style="font-size: 0.8em;">/opt/app.jar</div> <div style="font-size: 0.8em;">/lib/libc</div> </div> <p>layer</p> </div> <div>+</div> <div style="text-align: center;">  <div style="border: 1px solid black; padding: 10px; width: 100px; height: 100px; margin: 0 auto; display: flex; flex-direction: column; justify-content: center; align-items: center;"> <div style="font-size: 0.8em;">{</div> <div style="font-size: 0.8em;">"manifests": {</div> <div style="font-size: 0.8em;">"platform": {</div> <div style="font-size: 0.8em;">"os": "linux",</div> <div style="font-size: 0.8em;">... </div> <div style="font-size: 0.8em;">}</div> </div> <p>image index</p> </div> <div>+</div> <div style="text-align: center;">  <div style="border: 1px solid black; padding: 10px; width: 100px; height: 100px; margin: 0 auto; display: flex; flex-direction: column; justify-content: center; align-items: center;"> <div style="font-size: 0.8em;">{</div> <div style="font-size: 0.8em;">... </div> <div style="font-size: 0.8em;">"config": {</div> <div style="font-size: 0.8em;">"Cmd": [</div> <div style="font-size: 0.8em;">"java", "-jar",</div> <div style="font-size: 0.8em;">"app.jar"</div> <div style="font-size: 0.8em;">],</div> <div style="font-size: 0.8em;">... </div> <div style="font-size: 0.8em;">}</div> </div> <p>config</p> </div> </div> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 277 1297 337">OCI Image Configuration</h2> <p data-bbox="653 391 1913 553">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="653 591 1661 623">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="632 656 1507 727">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>





Claim 1	Accused Instrumentalities
	<p data-bbox="661 284 745 316">Layer</p> <ul data-bbox="688 357 1921 706" style="list-style-type: none"> • Image filesystems are composed of <i>layers</i>. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="661 755 856 787">Image JSON</p> <ul data-bbox="688 828 1921 1177" style="list-style-type: none"> • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="634 1201 1501 1274">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>Docker images are made up of layers, and each layer corresponds to a version of the image. Whenever a developer makes changes to the image, a new top layer is created, and this top layer replaces the previous top layer as the current version of the image. Previous layers are saved for rollbacks or to be re-used in other projects.</p> <p>Each time a container is created from a Docker image, yet another new layer called the container layer is created. Changes made to the container—such as the addition or deletion of files—are saved to the container layer only and exist only while the container is running. This iterative image-creation process enables increased overall efficiency since multiple live container instances can run from just a single base image, and when they do so, they leverage a common stack.</p> <p>https://www.ibm.com/topics/docker</p>



Claim 1	Accused Instrumentalities
	<p>The GNU C Library, commonly known as glibc, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system.</p> <p>https://en.wikipedia.org/wiki/Glibc</p>
<p>[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,</p>	<p>In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.</p> <p>For example, a base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and dependencies, including system libraries or shared library critical system elements (SLCSEs). When the container runs the image, it creates a runtime instance of that container image.</p> <p><i>See, e.g.:</i></p> <p>A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p>An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p>https://cloud.ibm.com/docs/containers?topic=containers-images</p>

Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>Docker images are made up of layers, and each layer corresponds to a version of the image. Whenever a developer makes changes to the image, a new top layer is created, and this top layer replaces the previous top layer as the current version of the image. Previous layers are saved for rollbacks or to be re-used in other projects.</p> <p>Each time a container is created from a Docker image, yet another new layer called the container layer is created. Changes made to the container—such as the addition or deletion of files—are saved to the container layer only and exist only while the container is running. This iterative image-creation process enables increased overall efficiency since multiple live container instances can run from just a single base image, and when they do so, they leverage a common stack.</p> <p>https://www.ibm.com/topics/docker</p> <p>Following software is installed on the Docker containers as part of the Product Master image deployment:</p> <ul style="list-style-type: none"> – Red Hat Enterprise Linux (RHEL) 7 Universal Base Image (UBI) base Docker image <p>https://www.ibm.com/docs/en/product-master/12.0.0?topic=deployment-installing-product-by-using-docker-images</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="646 277 1892 496">ubuntu <div data-bbox="1690 316 1837 332">1B+ · ☆10K+</div><div data-bbox="783 354 961 373">Updated 15 days ago</div><div data-bbox="783 391 1413 410">Ubuntu is a Debian-based Linux operating system based on free software.</div><div data-bbox="783 436 1360 456">Linux IBM Z 386 riscv64 x86-64 ARM ARM 64 PowerPC 64 LE</div></div> <div data-bbox="646 537 1892 756">debian <div data-bbox="1707 576 1854 592">1B+ · ☆4.9K</div><div data-bbox="783 618 993 638">Updated 35 minutes ago</div><div data-bbox="783 656 1549 675">Debian is a Linux distribution that's composed entirely of free and open-source software.</div><div data-bbox="783 703 1455 722">Linux riscv64 x86-64 ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z</div></div> <div data-bbox="636 803 1533 836">https://hub.docker.com/search?image_filter=official&type=image&q=</div>

Claim 1	Accused Instrumentalities																																																						
	<table><tr><th>Platform</th><th>x86_64 / amd64</th><th>arm64 / aarch64</th><th>arm (32-bit)</th><th>ppc64le</th><th>s390x</th></tr><tr><td>CentOS</td><td>✓</td><td>✓</td><td></td><td>✓</td><td></td></tr><tr><td>Debian</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td></td></tr><tr><td>Fedora</td><td>✓</td><td>✓</td><td></td><td>✓</td><td></td></tr><tr><td>Raspberry Pi OS (32-bit)</td><td></td><td></td><td>✓</td><td></td><td></td></tr><tr><td>RHEL (s390x)</td><td></td><td></td><td></td><td></td><td>✓</td></tr><tr><td>SLES</td><td></td><td></td><td></td><td></td><td>✓</td></tr><tr><td>Ubuntu</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td><td>✓</td></tr><tr><td>Binaries</td><td>✓</td><td>✓</td><td>✓</td><td></td><td></td></tr></table> <p>https://docs.docker.com/engine/install/</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p>	Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x	CentOS	✓	✓		✓		Debian	✓	✓	✓	✓		Fedora	✓	✓		✓		Raspberry Pi OS (32-bit)			✓			RHEL (s390x)					✓	SLES					✓	Ubuntu	✓	✓	✓	✓	✓	Binaries	✓	✓	✓		
Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x																																																		
CentOS	✓	✓		✓																																																			
Debian	✓	✓	✓	✓																																																			
Fedora	✓	✓		✓																																																			
Raspberry Pi OS (32-bit)			✓																																																				
RHEL (s390x)					✓																																																		
SLES					✓																																																		
Ubuntu	✓	✓	✓	✓	✓																																																		
Binaries	✓	✓	✓																																																				

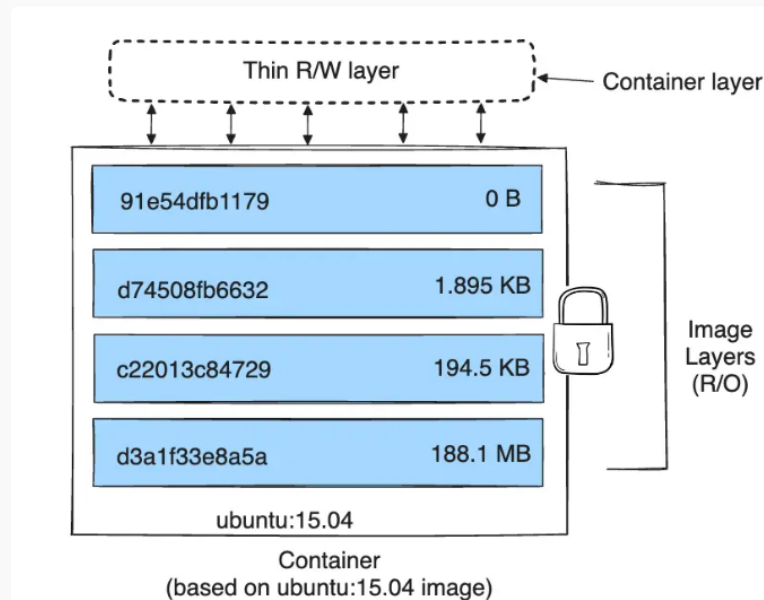
Claim 1	Accused Instrumentalities
	<p data-bbox="892 293 1587 329">A container is a runtime instance of a docker image.</p> <p data-bbox="892 386 1304 418">A Docker container consists of</p> <div data-bbox="659 475 1350 651"><p data-bbox="659 529 789 561">container</p><ul style="list-style-type: none"><li data-bbox="909 475 1163 508">• A Docker image<li data-bbox="909 548 1304 581">• An execution environment<li data-bbox="909 621 1350 654">• A standard set of instructions</div> <p data-bbox="636 711 1157 747">https://docs.docker.com/glossary/#image</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="646 280 1272 349">About storage drivers</h2> <p data-bbox="646 394 1871 521">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="646 589 1564 646">Storage drivers versus Docker volumes</h2> <p data-bbox="646 683 1913 950">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="646 1000 1902 1127">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1339 1047 1535 1076" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="636 1154 1224 1187"><a data-bbox="636 1154 1224 1187" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="657 277 1081 334">Images and layers</h2> <p data-bbox="657 370 1822 448">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="657 488 1908 852"> # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py </pre> <p data-bbox="657 894 1900 1203">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="634 1224 1226 1256">https://docs.docker.com/storage/storagedriver/</p>

Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the [Best practices for writing Dockerfiles](#) and [use multi-stage builds](#) sections to learn how to optimize your Dockerfiles for efficient images.

The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.

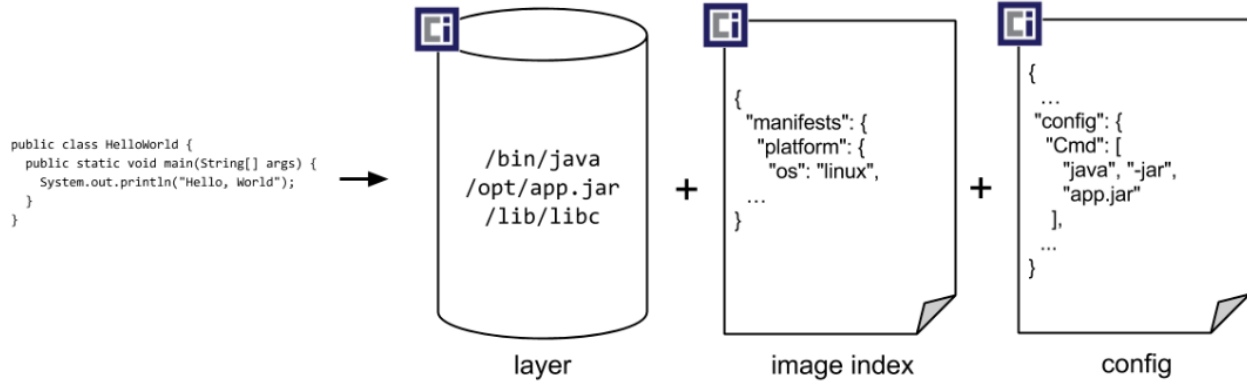





<https://docs.docker.com/storage/storagedriver/>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 289 919 354">Volumes</h2> <p data-bbox="653 407 1906 537">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="636 558 1308 591">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="653 643 1226 691">Container environment</h2> <p data-bbox="653 729 1474 797">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="695 834 1451 992" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="636 1029 1528 1062">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="659 282 877 345">Images</h2> <p data-bbox="659 378 1522 532">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="659 565 1528 641">You typically create a container image of your application and push it to a registry before referring to it in a <u>Pod</u>.</p> <p data-bbox="634 667 1329 699">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="653 743 919 800">Volumes</h2> <p data-bbox="653 841 1482 917">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers.</p> <p data-bbox="653 922 1430 954">One problem occurs when a container crashes or is stopped.</p> <p data-bbox="653 964 1528 1284">Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <u>Pod</u> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <u>volume</u> abstraction solves both of these problems. Familiarity with <u>Pods</u> is suggested.</p> <p data-bbox="634 1310 1308 1343">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="659 289 1299 349"><h2>Open Container Initiative</h2><hr/></div> <div data-bbox="659 410 1186 456"><h3>Image Format Specification</h3><hr/></div> <div data-bbox="659 503 1904 578"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="659 612 1908 686"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="625 712 1482 784"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

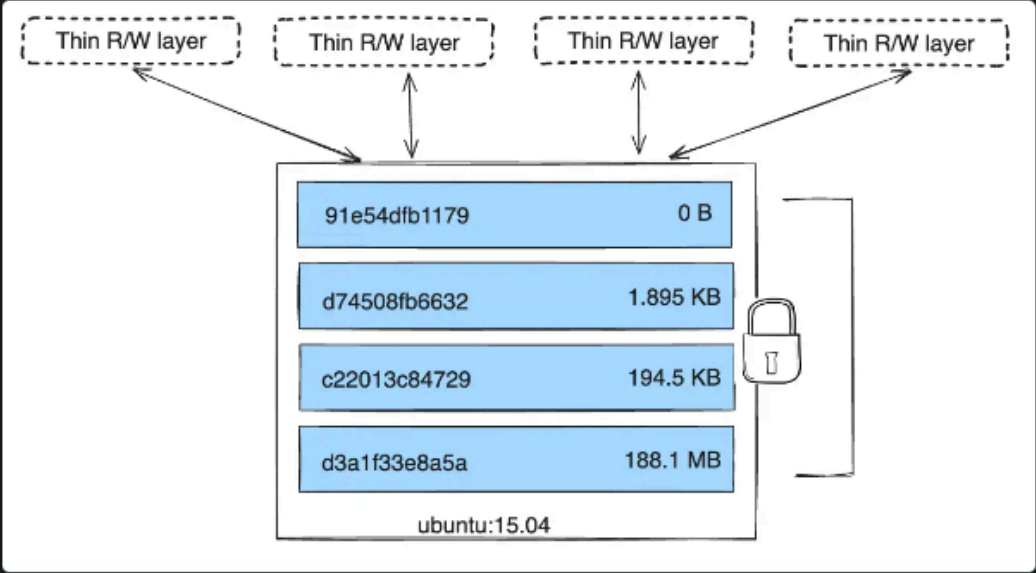
Claim 1	Accused Instrumentalities
	<p>Overview</p> <p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p>  <pre> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p>→</p> <div style="display: flex; align-items: center; justify-content: space-around;"> <div style="text-align: center;">  <div style="border: 1px solid black; border-radius: 50%; padding: 10px; width: 100px; height: 100px; margin: 0 auto; display: flex; flex-direction: column; justify-content: center; align-items: center;"> <div style="font-size: 0.8em;">/bin/java</div> <div style="font-size: 0.8em;">/opt/app.jar</div> <div style="font-size: 0.8em;">/lib/libc</div> </div> <p>layer</p> </div> <div>+</div> <div style="text-align: center;">  <div style="border: 1px solid black; padding: 10px; width: 100px; height: 100px; margin: 0 auto; display: flex; flex-direction: column; justify-content: center; align-items: center;"> <div style="font-size: 0.8em;">{</div> <div style="font-size: 0.8em;">"manifests": {</div> <div style="font-size: 0.8em;">"platform": {</div> <div style="font-size: 0.8em;">"os": "linux",</div> <div style="font-size: 0.8em;">... </div> <div style="font-size: 0.8em;">}</div> </div> <p>image index</p> </div> <div>+</div> <div style="text-align: center;">  <div style="border: 1px solid black; padding: 10px; width: 100px; height: 100px; margin: 0 auto; display: flex; flex-direction: column; justify-content: center; align-items: center;"> <div style="font-size: 0.8em;">{</div> <div style="font-size: 0.8em;">... </div> <div style="font-size: 0.8em;">"config": {</div> <div style="font-size: 0.8em;">"Cmd": [</div> <div style="font-size: 0.8em;">"java", "-jar",</div> <div style="font-size: 0.8em;">"app.jar"</div> <div style="font-size: 0.8em;">],</div> <div style="font-size: 0.8em;">... </div> <div style="font-size: 0.8em;">}</div> </div> <p>config</p> </div> </div> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="653 277 1297 337">OCI Image Configuration</h2> <p data-bbox="653 391 1913 553">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="653 591 1661 623">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="632 656 1507 727">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

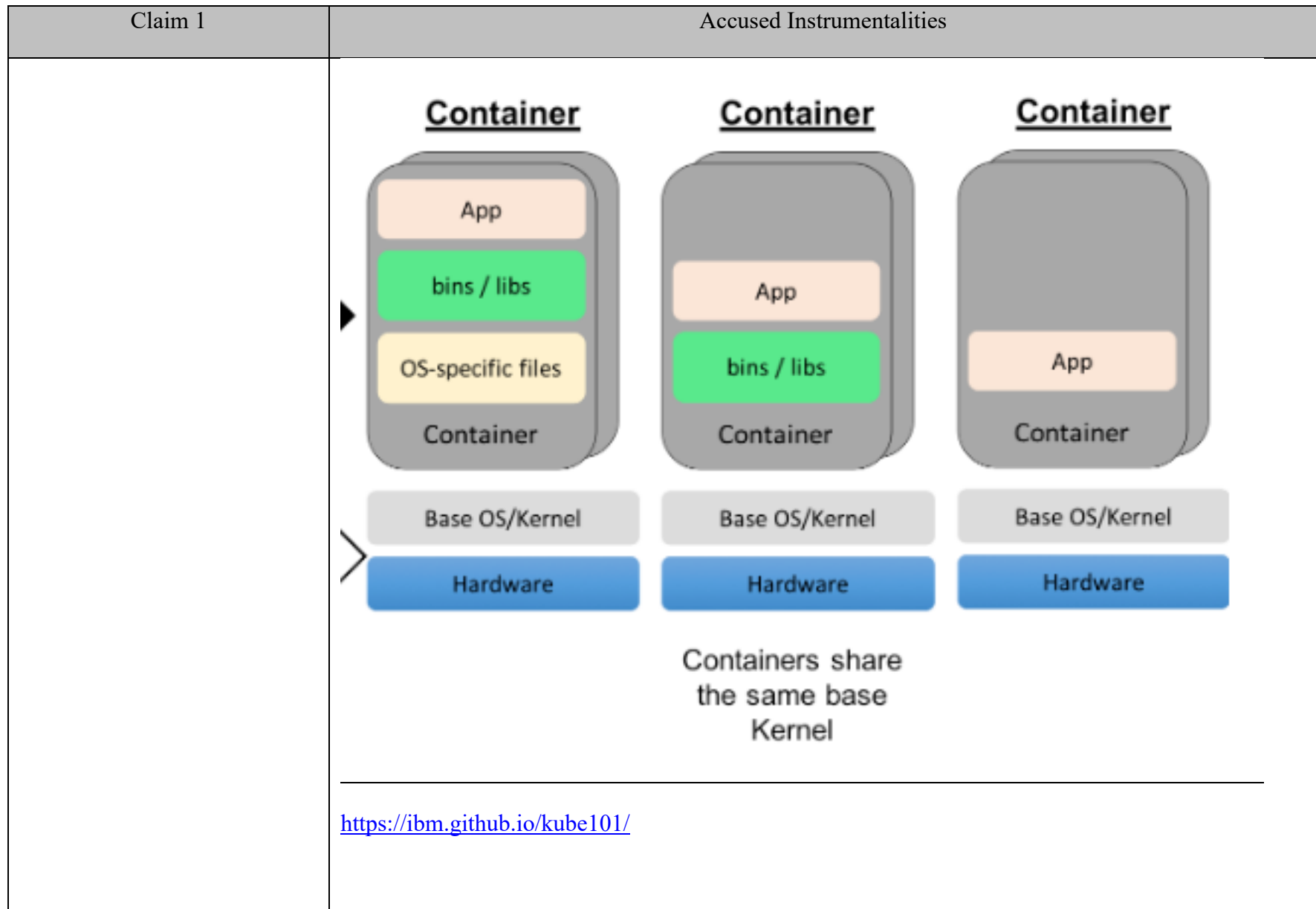
Claim 1	Accused Instrumentalities
	<p data-bbox="661 284 745 321">Layer</p> <ul data-bbox="688 358 1919 706" style="list-style-type: none"> • Image filesystems are composed of <i>layers</i>. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="661 755 856 792">Image JSON</p> <ul data-bbox="688 829 1919 1177" style="list-style-type: none"> • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="634 1203 1503 1274">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
<p>[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and</p>	<p>In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.</p> <p>When a Docker or Kubernetes image is used to create a container, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications running in their respective containers. The image includes essential system files, libraries, and dependencies required to run the software application within the container. The containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it operates in isolation and runs its own instance of the software application without sharing resources or critical system elements with other containers. This ensures that each container has its own isolated context. Docker or Kubernetes containers can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same or a different function.</p> <p><i>See, e.g.:</i></p>

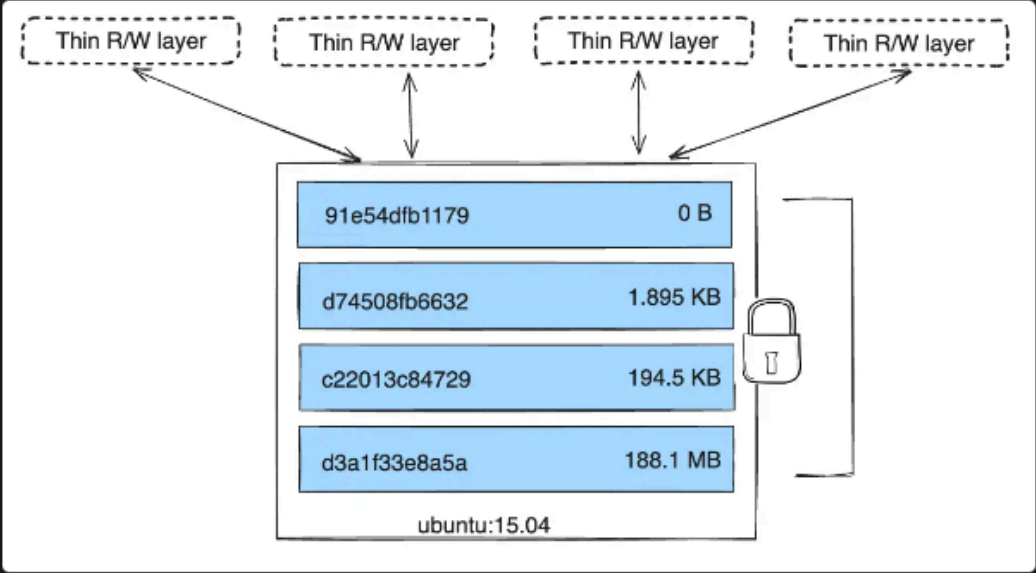
Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>It's possible to build a Docker image from scratch, but most developers pull them down from common repositories. Multiple Docker images can be created from a single base image, and they'll share the commonalities of their stack.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="653 277 1919 402">Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p data-bbox="632 1101 1226 1133">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p>



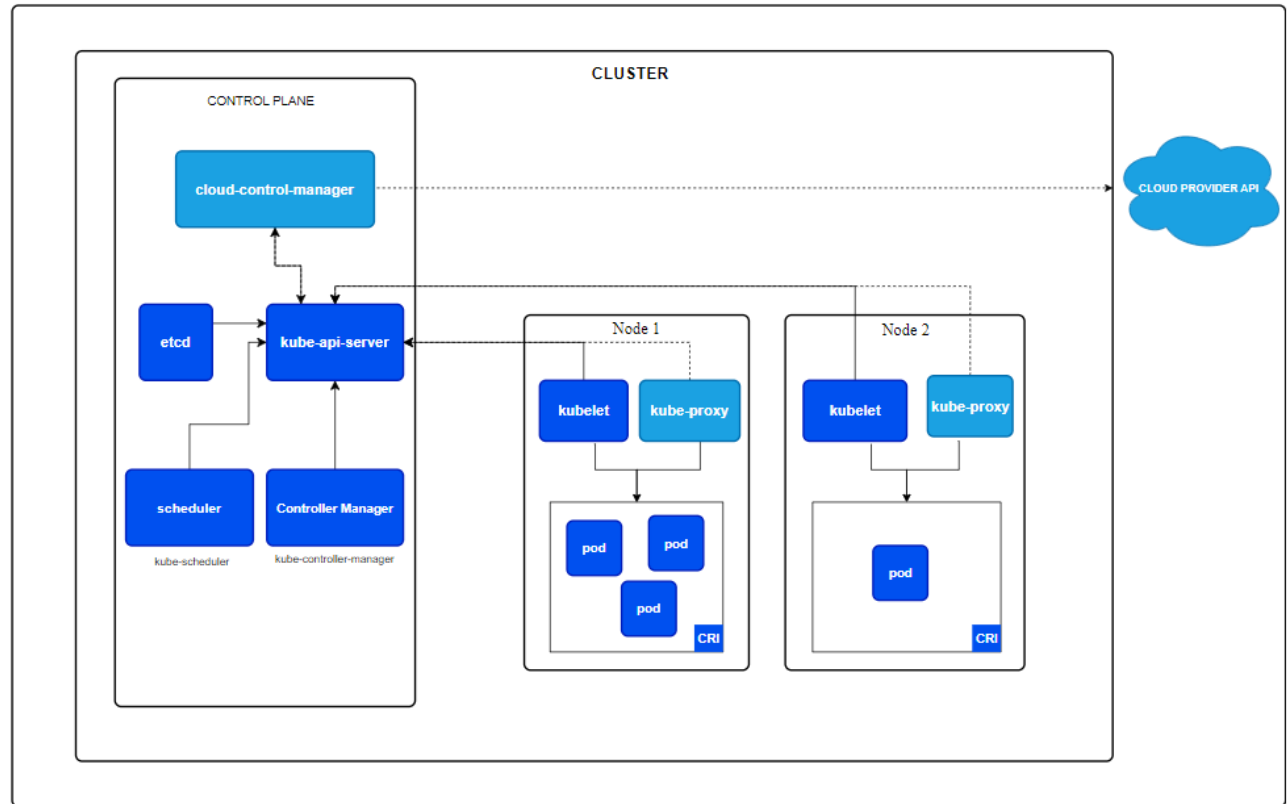
Claim 1	Accused Instrumentalities
<p>[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p>	<p>In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p> <p>For example, in Docker or Kubernetes containers, each container operates independently, and a base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When an image is used to create a container in the Accused Instrumentality, an instance of the SLCSE is provided to a software application. Therefore, different instances of the SLCSE are provided to different applications for performing either a same or a different function, simultaneously.</p> <p><i>See, e.g.:</i></p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <p><i>Docker images</i> contain executable application source code as well as all the tools, libraries, and dependencies that the application code needs to run as a container. When you run the Docker image, it becomes one instance (or multiple instances) of the container.</p> <p>https://www.ibm.com/topics/docker</p>

Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p> <p>A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.</p>

Claim 1	Accused Instrumentalities
	https://docs.docker.com/get-started/overview/

Claim 2

Claim 2	Accused Instrumentalities
2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.	<p>Each Accused Instrumentality comprises or constitutes a computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.</p> <p>For example, an individual host/node runs multiple containers and/or pods simultaneously, each of which has an instance of an SLCSE.</p> <p><i>See, e.g.:</i></p>

Claim 2**Accused Instrumentalities**

Kubernetes cluster architecture

<https://kubernetes.io/docs/concepts/architecture/>






Claim 2	Accused Instrumentalities
	<h1 data-bbox="661 261 1136 337">Containers</h1> <p data-bbox="661 399 1919 553">Each container that you run is repeatable; the standardization from having dependencies included means that you get the same behavior wherever you run it.</p> <p data-bbox="661 613 1919 768">Containers decouple applications from the underlying host infrastructure. This makes deployment easier in different cloud or OS environments.</p> <p data-bbox="661 828 1892 982">Each <u>node</u> in a Kubernetes cluster runs the containers that form the Pods assigned to that node. Containers in a Pod are co-located and co-scheduled to run on the same node.</p> <p data-bbox="644 1039 1241 1068">https://kubernetes.io/docs/concepts/containers/</p>

Claim 2	Accused Instrumentalities
	<h1 data-bbox="674 248 1520 318">Kubernetes Scheduler</h1> <p data-bbox="674 370 1650 456">In Kubernetes, <i>scheduling</i> refers to making sure that <u>Pods</u> are matched to <u>Nodes</u> so that <u>Kubelet</u> can run them.</p> <h2 data-bbox="674 581 1325 651">Scheduling overview</h2> <p data-bbox="674 695 1745 935">A scheduler watches for newly created Pods that have no Node assigned. For every Pod that the scheduler discovers, the scheduler becomes responsible for finding the best Node for that Pod to run on. The scheduler reaches this placement decision taking into account the scheduling principles described below.</p> <p data-bbox="674 987 1696 1130">If you want to understand why Pods are placed onto a particular Node, or if you're planning to implement a custom scheduler yourself, this page will help you learn about scheduling.</p> <p data-bbox="642 1182 1566 1211">https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/</p>

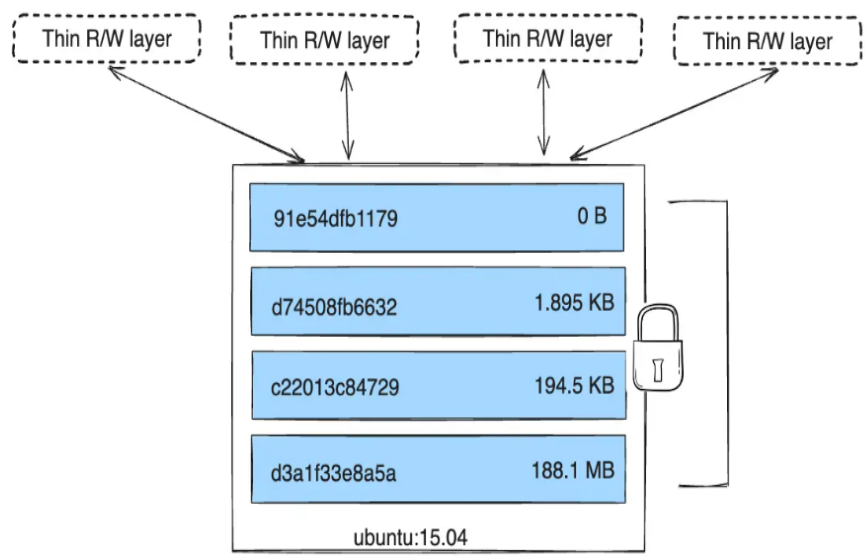
Claim 2	Accused Instrumentalities
	<h2 data-bbox="657 250 1220 315">Running containers</h2> <p data-bbox="657 363 1919 487">Docker runs processes in isolated containers. A container is a process which runs on a host. The host may be local or remote. When you execute <code>docker run</code>, the container process that runs is isolated in that it has its own file system, its own networking, and its own isolated process tree separate from the host.</p> <p data-bbox="646 526 1230 558">https://docs.docker.com/engine/reference/run/</p>

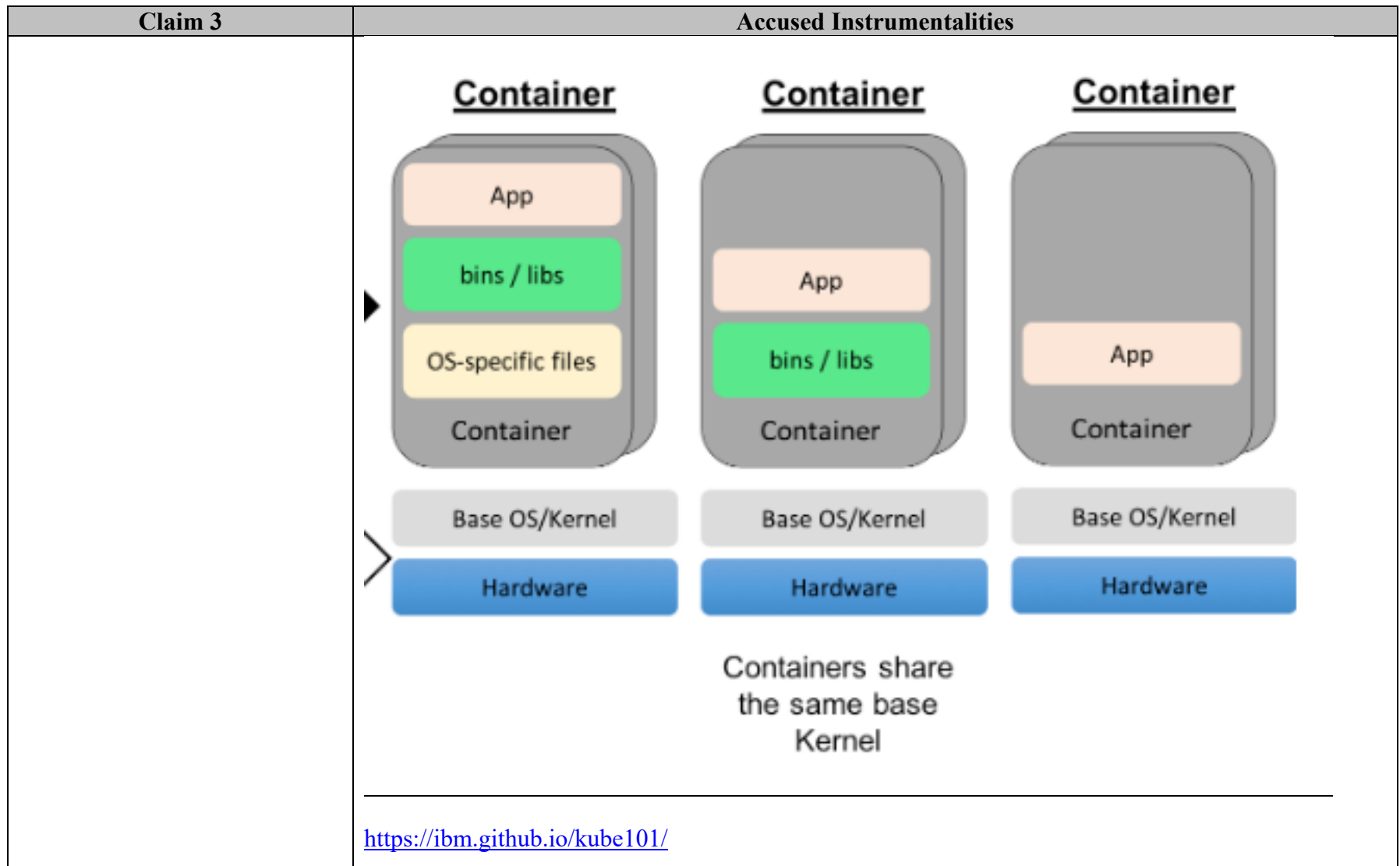
Claim 3

Claim 3	Accused Instrumentalities
<p data-bbox="205 732 611 948">3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.</p>	<p data-bbox="646 732 1948 839">Each Accused Instrumentality comprises or constitutes a computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.</p> <p data-bbox="646 862 1934 932">For example, both Docker and Kubernetes systems preserve the host kernel substantially unchanged; therefore the OSCSEs corresponding to the SLCSEs remain in the operating system kernel.</p> <p data-bbox="646 954 758 992"><i>See, e.g.:</i></p> <p data-bbox="674 1040 1730 1068">A Docker image is the basis for every container that you create with IBM Cloud® Kubernetes Service.</p> <p data-bbox="674 1105 1902 1213">An image is created from a Dockerfile, which is a file that contains instructions to build the image. A Dockerfile might reference build artifacts in its instructions that are stored separately, such as an app, the app's configuration, and its dependencies.</p> <p data-bbox="646 1279 1457 1312">https://cloud.ibm.com/docs/containers?topic=containers-images</p>

Claim 3	Accused Instrumentalities		
	Docker base image	Supported versions	Source of security notices
	Alpine	All stable versions with vendor security support.	Alpine SecDB database  .
	Debian	All stable versions with vendor security support. CVEs on binary packages that are associated with the Debian source package <code>linux</code> , such as <code>linux-libc-dev</code> , are not reported. Most of these binary packages are kernel and kernel modules, which are not run in container images.	Debian Security Bug Tracker  .
	GoogleContainerTools distroless	All stable versions with vendor security support.	GoogleContainerTools distroless  .
	Red Hat® Enterprise Linux® (RHEL)	RHEL/UBI 7, RHEL/UBI 8, and RHEL/UBI 9	Red Hat Security Data API  .
	Ubuntu	All stable versions with vendor security support.	Ubuntu CVE Tracker  .
	https://cloud.ibm.com/docs/Registry?topic=Registry-va_index&interface=ui		

Claim 3	Accused Instrumentalities
	<p data-bbox="682 240 1045 289">Container images</p> <p data-bbox="682 316 1377 440">A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <hr data-bbox="646 459 1461 462"/> <p data-bbox="646 467 1241 500">https://kubernetes.io/docs/concepts/containers/</p> <p data-bbox="653 581 1551 764">Container image files are complete, static and executable versions of an application or service and differ from one technology to another. Docker images are made up of multiple layers, which start with a base image that includes all of the dependencies needed to execute code in a container. Each image has a readable/writable layer on top of static unchanging layers. Because each container has its own specific container layer that customizes that specific container, underlying image layers can be saved and reused in multiple containers. An Open Container Initiative (OCI)</p> <p data-bbox="646 776 1892 846">https://www.techtarget.com/searchitoperations/definition/container-containerization-or-container-based-virtualization</p>

Claim 3	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>



Claim 4

Claim 4	Accused Instrumentalities
<p>4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.</p>	<p>Each Accused Instrumentality comprises or constitutes a computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.</p> <p>For example, the SLCSEs in a container use system calls to access services in the operating system kernel. For example, the glibc library (or other similar library) in the container uses system calls to interface with the host Linux kernel. In general, system calls can be observed using a tool such as strace.</p> <p><i>See, e.g.:</i></p> <p>The GNU C Library, commonly known as glibc, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system.</p> <p>https://en.wikipedia.org/wiki/Glibc</p>

We can now get the process id directly from the cgroup. It will be located in the cgroup.procs file.

Terminal 2 - Worker Node

Get the process id

```
$ cat cri-containerd-ceeeef06afe89c8223d33b11e8d9e0b207118ac4dac3af826687668ee1ee
16254
```

Validate what is running under the process

```
$ ps aux | grep 16254
```

```
azureus+ 16254 0.0 0.1 713972 10476 ? Ssl 15:04 0:00 ./faultyapp
azureus+ 94806 0.0 0.0 7004 2168 pts/0 S+ 16:22 0:00 grep --color=a
```

Got it! With that, we can try to find out what is going out inside the app. Lets try to run strace to get some more insight.

Terminal 2 - Worker Node

```
$ sudo strace -p 16254 -f
```

```
...
```

The app is trying to read a file port.txt

```
[pid 16269] openat(AT_FDCWD, "port.txt", O_RDONLY|O_CLOEXEC <unfinished ...>
```

```
[pid 16254] epoll_pwait(5, <unfinished ...>
```

The file does not exist

```
[pid 16269] <... openat resumed> = -1 ENOENT (No such file or directory)
```

```
[pid 16254] <... epoll_pwait resumed>[, 128, 0, NULL, 0) = 0
```

```
[pid 16269] write(1, "Something went wrong...\n", 24 <unfinished ...>
```

After filtering the output, we can see the application is trying to read a text file called port.txt, and a few lines later, there is a message stating ENOENT (No such file or directory). Let's create that file.

<https://www.berops.com/blog/a-different-method-to-debug-kubernetes-pods>

Claim 18

Claim 18	Accused Instrumentalities
<p>18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.</p>	<p>Each Accused Instrumentality comprises or constitutes a computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.</p> <p>For example, in a typical case the SLCSEs come from a Linux distribution independent of the host operating system, and thus are not identical to the OSCSEs. For another example, the SLCSEs are provided to the computer system through a separate process than the process by which the OSCSEs are provided to the computer system, and thus are not copied from the OSCSEs.</p> <p><i>See, e.g.:</i></p> <p>Containers are executable units of software in which application code is packaged along with its libraries and dependencies, in common ways so that the code can be run anywhere—whether it be on desktop, traditional IT or the cloud.</p> <p>To do this, containers take advantage of a form of operating system (OS) virtualization in which features of the OS kernel (e.g. Linux namespaces and cgroups, Windows silos and job objects) can be leveraged to isolate processes and control the amount of CPU, memory and disk that those processes can access.</p> <p>Containers are small, fast and portable because unlike a virtual machine, containers do not need to include a guest OS in every instance and can instead simply leverage the features and resources of the host OS.</p> <p>https://www.ibm.com/topics/containers</p>

Claim 18	Accused Instrumentalities
	<p>Containers use a form of operating system (OS) virtualization. Put simply, they leverage features of the host operating system to isolate processes and control the processes' access to CPUs, memory and desk space.</p> <p>https://www.ibm.com/blog/containers-vs-vms/</p> <h2>Container images</h2> <p>A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p>https://kubernetes.io/docs/concepts/containers/</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p>

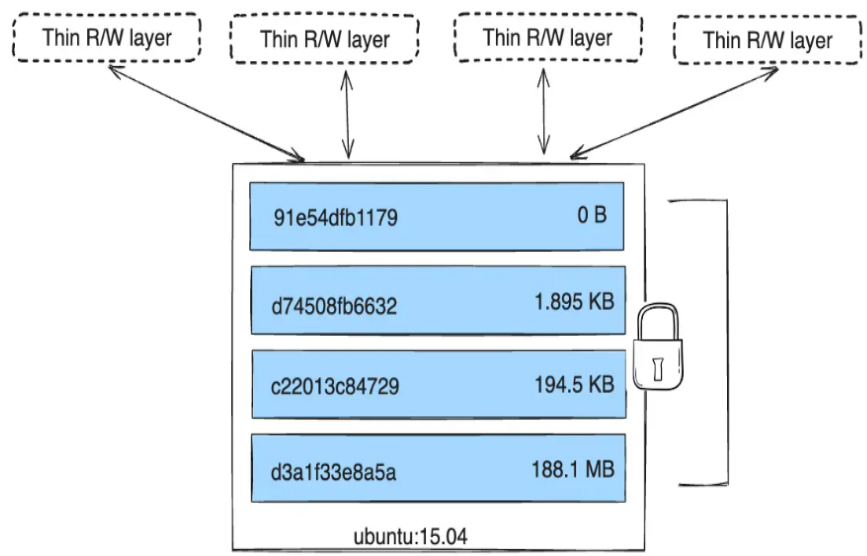
Claim 18	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>

EXHIBIT M

**UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION**

VIRTAMOVE, CORP.,

Plaintiff,

v.

MICROSOFT CORPORATION,
Defendant.

Case No. 7:24-cv-00338

JURY TRIAL DEMANDED

**COMPLAINT FOR PATENT INFRINGEMENT AGAINST
MICROSOFT CORPORATION**

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, “Plaintiff” or “VirtaMove”) makes the following allegations against Defendant Microsoft Corporation (“Defendant” or “Microsoft”):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant’s unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058 (collectively, the “Asserted Patents”). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as Appzero Software Corp. (“Appzero”), which was established in 2010.

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repackage, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove's mission is to move and modernize the world's server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach \$2.1 billion in 2019 and \$4.3 billion in 2022—a compound annual growth rate (“CAGR”) of 30%. *See, e.g.*, <https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022>. Another source reported the application containers market had a market size of \$5.45 billion in 2024 and estimated it to reach \$19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.*, <https://www.mordorintelligence.com/industry-reports/application-container-market>.

6. Microsoft is a Washington corporation with a principal place of business located at 1 Microsoft Way, Redmond, Washington 98052-8300. Microsoft is registered to do business in Texas and can be served via its registered agent, Corporation Service Company dba CSC – Lawyers Incorporating Service Company at 211 East 7th Street, Suite 620, Austin, Texas 78701-

3218. Microsoft maintains a permanent physical presence within the Western District of Texas, conducting business from at least its locations at: 10900 Stonelake Boulevard, Suite 225, Austin, Texas 78759; Concord Park II 401 East Sonterra Boulevard, Suite 300, San Antonio, Texas 78258; as well as other locations in and around the Austin and San Antonio areas.

JURISDICTION AND VENUE

7. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

9. Venue is proper in this District because Defendant resides in this District, has a regular and established place of business in this District, and has committed acts of infringement within this District.

COUNT I

INFRINGEMENT OF U.S. PATENT NO. 7,519,814

10. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

11. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 ('814

patent), titled “System for Containerization of Application Sets,” issued on April 14, 2009. A true and correct copy of the ’814 Patent is attached as **Exhibit 1**.

12. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products (“Accused Products”), such as, e.g., Azure Kubernetes Service (“AKS”), that directly infringe, literally and/or under the doctrine of equivalents, claims of the ’814 patent. The infringement of the Asserted Patents is also attributable to Defendant. Defendant directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

13. Defendant also knowingly and intentionally induces infringement of claims of the ’814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the ’814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and served on Defendant. Despite this knowledge of the ’814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the ’814 patent. Defendant does so, knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the ’814 patent, thereby specifically intending for and inducing its customers to infringe the ’814 patent through the customers’ normal and customary use of the Accused Products.

14. Defendant has also infringed, and continue to infringe, claims of the ’814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent,

and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '814 patent, in violation of 35 U.S.C. § 271(c).

15. The Accused Products satisfy all claim limitations of one or more claims of the '814 patent. A claim chart comparing independent claim 1 of the '814 patent to a representative Accused Product is attached as **Exhibit 2**, which is hereby incorporated by reference in its entirety.

16. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the '814 patent pursuant to 35 U.S.C. § 271.

17. As a result of Defendant's infringement of the '814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. VirtaMove has complied with the requirements of 35 U.S.C. § 287 and is not aware of any unmarked products that practice the claims of the '814 patent. In the alternative, either VirtaMove's product was marked before the filing of this lawsuit, or no requirement for marking applies.

COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

18. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

19. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 ('058 patent), titled "Computing System Having User Mode Critical System Elements as Shared Libraries," issued on August 24, 2010. A true and correct copy of the '058 patent is attached as **Exhibit 3**.

20. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., Azure Kubernetes Service ("AKS"), that directly infringe, literally and/or under the doctrine of equivalents. The infringement of the Asserted Patents is also attributable to Defendant. Defendant directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

21. Defendant also knowingly and intentionally induces infringement of claims of the '058 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and served. Despite this knowledge of the '058 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '058 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '058 patent, thereby specifically intending for and inducing its customers to infringe the '058 patent through the customers' normal and customary use of the Accused Products.

22. Defendant has also infringed, and continue to infringe, claims of the '058 patent by

offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '058 patent, in violation of 35 U.S.C. § 271(c).

23. The Accused Products satisfy all claim limitations of one or more claims of the '058 patent. A claim chart comparing claim 1 of the '058 patent to a representative Accused Product is attached as **Exhibit 4**, which is hereby incorporated by reference in its entirety.

24. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the '058 patent pursuant to 35 U.S.C. § 271.

25. As a result of Defendant's infringement of the '058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. VirtaMove has complied with the requirements of 35 U.S.C. § 287. 35 U.S.C. § 287 requires the marking of a "patented articles," however VirtaMove makes, offers for sale, and/or sells software and software support services. VirtaMove is unaware of instances where it made, offered for sale, and/or sold "a processor," as required by claim 1 of the '058 patent, with these products. Thus, VirtaMove is not required to mark any of these products that it makes, offers for sale, and/or sells. In the alternative, to the extent VirtaMove is found to have "made," "offered for sale," and/or

“sold” “patented articles,” including to the extent VirtaMove is found to have made, offered for sale, and/or sold products with “a processor” as required by claim 1 of the ’058 patent, VirtaMove’s products were marked before the filing of this lawsuit.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

- a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;
- b. A permanent injunction prohibiting Defendant from further acts of infringement of each of the ’814 and ’058 patents;
- c. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant’s infringement of each of the Asserted Patents;
- d. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;
- e. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys’ fees against Defendant; and
- f. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury of any issues so triable by right.

Dated: December 20, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)

rmirzaie@raklaw.com

Marc A. Fenster (CA SBN 181067)

mfenster@raklaw.com

Neil A. Rubin (CA SBN 250761)

nrubin@raklaw.com

Amy E. Hayden (CA SBN 287026)

ahayden@raklaw.com

Jacob R. Buczek (CA SBN 269408)

jbuczek@raklaw.com

James S. Tsuei (CA SBN 285530)

jtsuei@raklaw.com

James A. Milkey (CA SBN 281283)

jmilkey@raklaw.com

Christian W. Conkle (CA SBN 306374)

cconkle@raklaw.com

Jonathan Ma (CA SBN 312773)

jma@raklaw.com

Daniel Kolko (CA SBN 341680)

dkolko@raklaw.com

Mackenzie Paladino (NY SBN 6039366)

mpaladino@raklaw.com

RUSS AUGUST & KABAT

12424 Wilshire Boulevard, 12th Floor

Los Angeles, CA 90025

Telephone: (310) 826-7474

Qi (Peter) Tong (TX SBN 24119042)

ptong@raklaw.com

RUSS AUGUST & KABAT

8080 N. Central Expy., Suite 1503

Dallas, TX 75206

Telephone: (310) 826-7474

Attorneys for Plaintiff VirtaMove, Corp.

Exhibit 1



US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,519,814 B2**
(45) **Date of Patent:** **Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

WO WO 2006/039181 A 4/2006

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248;
709/214; 719/319

(58) **Field of Classification Search** 713/167;
719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

OTHER PUBLICATIONS

Soltész, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667.*

Primary Examiner—Kambiz Zand

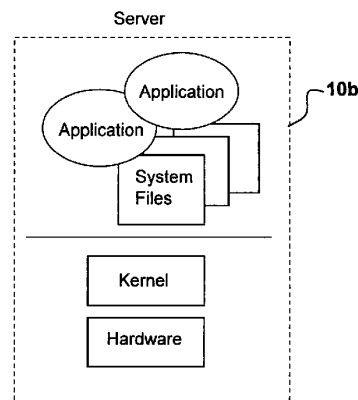
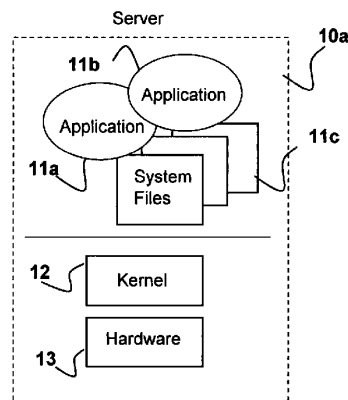
Assistant Examiner—Ronald Baum

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



US 7,519,814 B2

Page 2

U.S. PATENT DOCUMENTS				2002/0004854 A1	1/2002	Hartley	
6,847,970 B2 *	1/2005	Keller et al.	707/100	2002/0174215 A1	11/2002	Schaefer	709/224
7,076,784 B1 *	7/2006	Russell et al.	719/315	2003/0101292 A1	5/2003	Fisher	
7,287,259 B2 *	10/2007	Grier et al.	719/331	* cited by examiner			

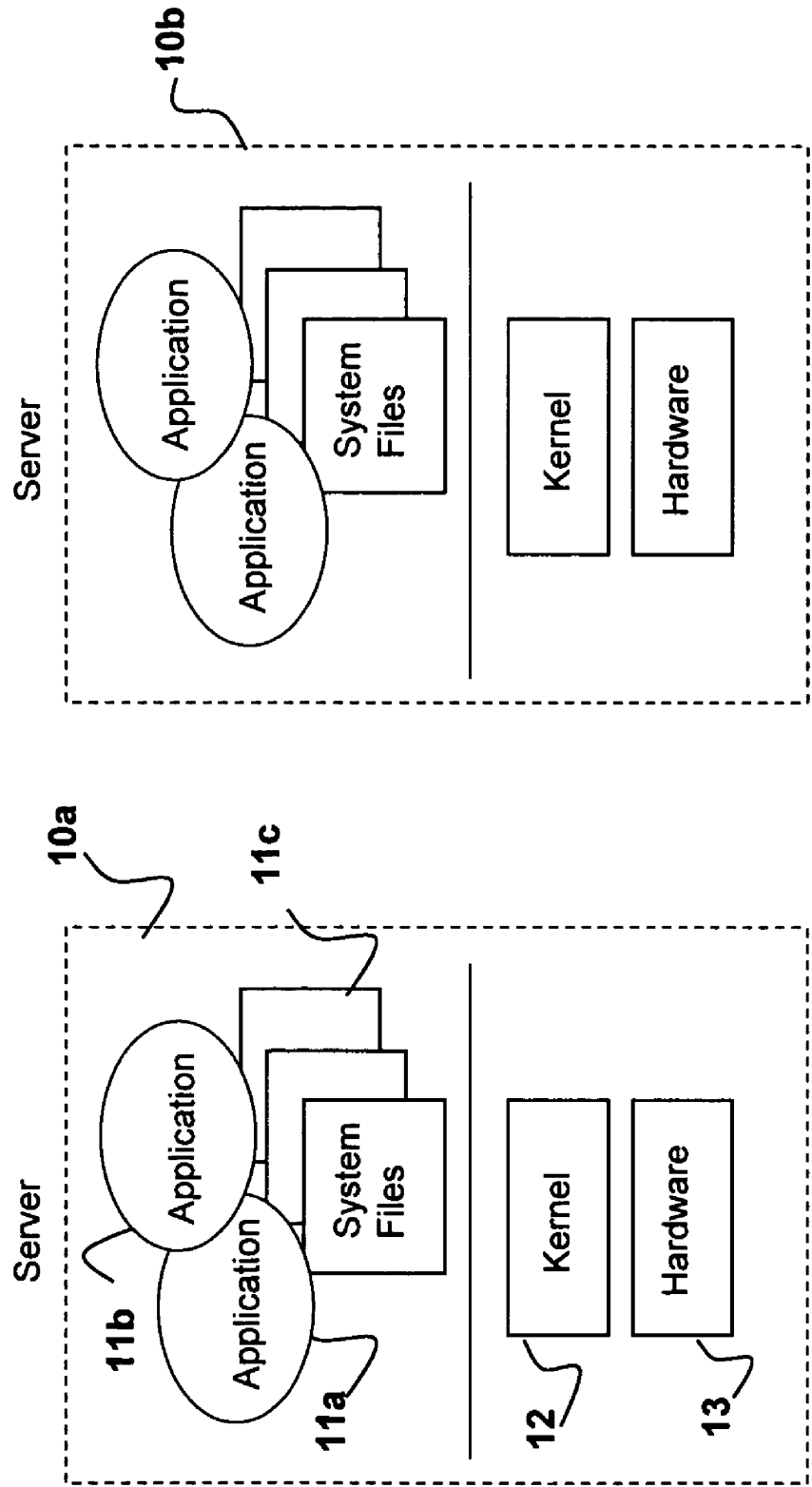


Figure 1

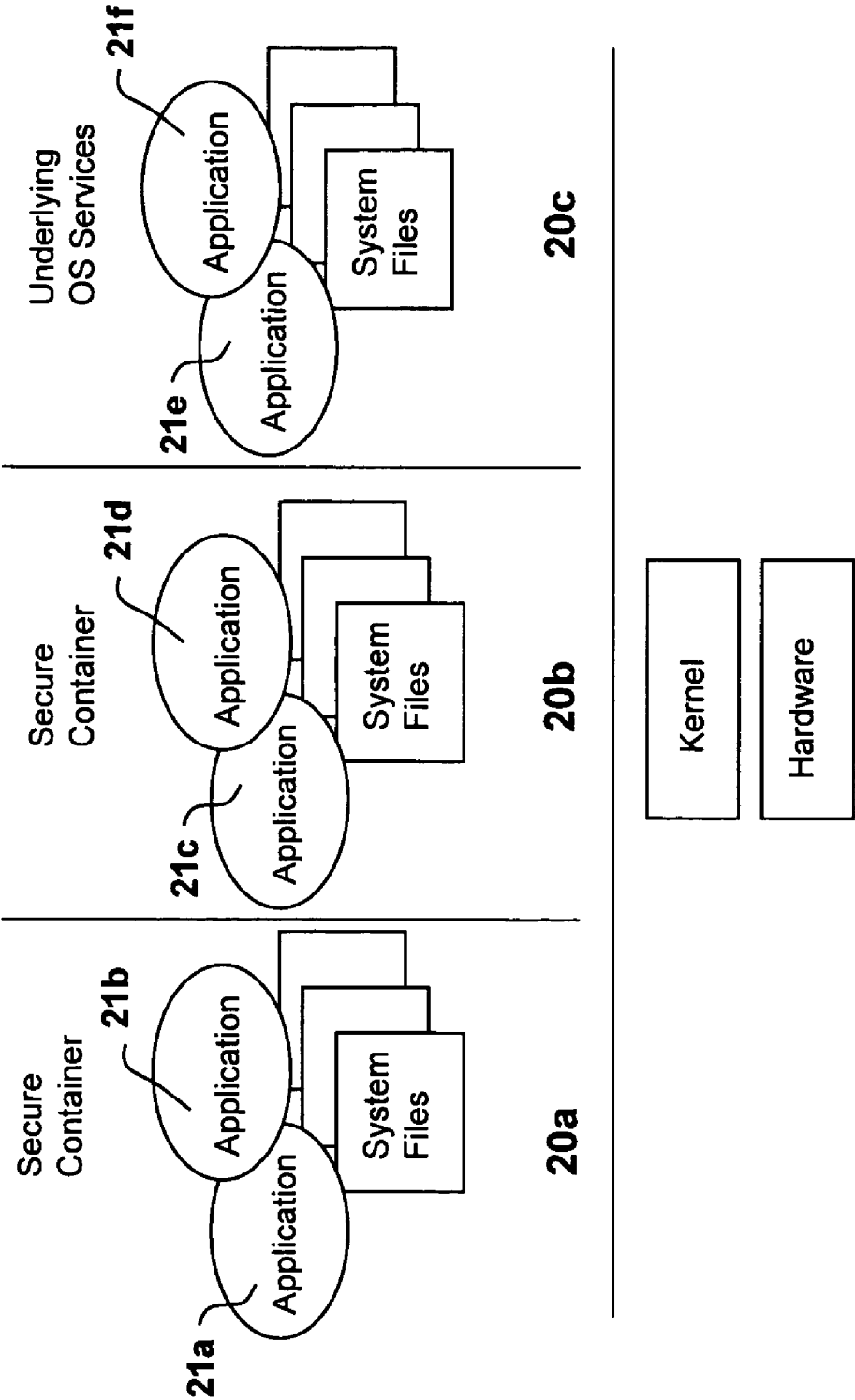


Figure 2

U.S. Patent

Apr. 14, 2009

Sheet 3 of 17

US 7,519,814 B2

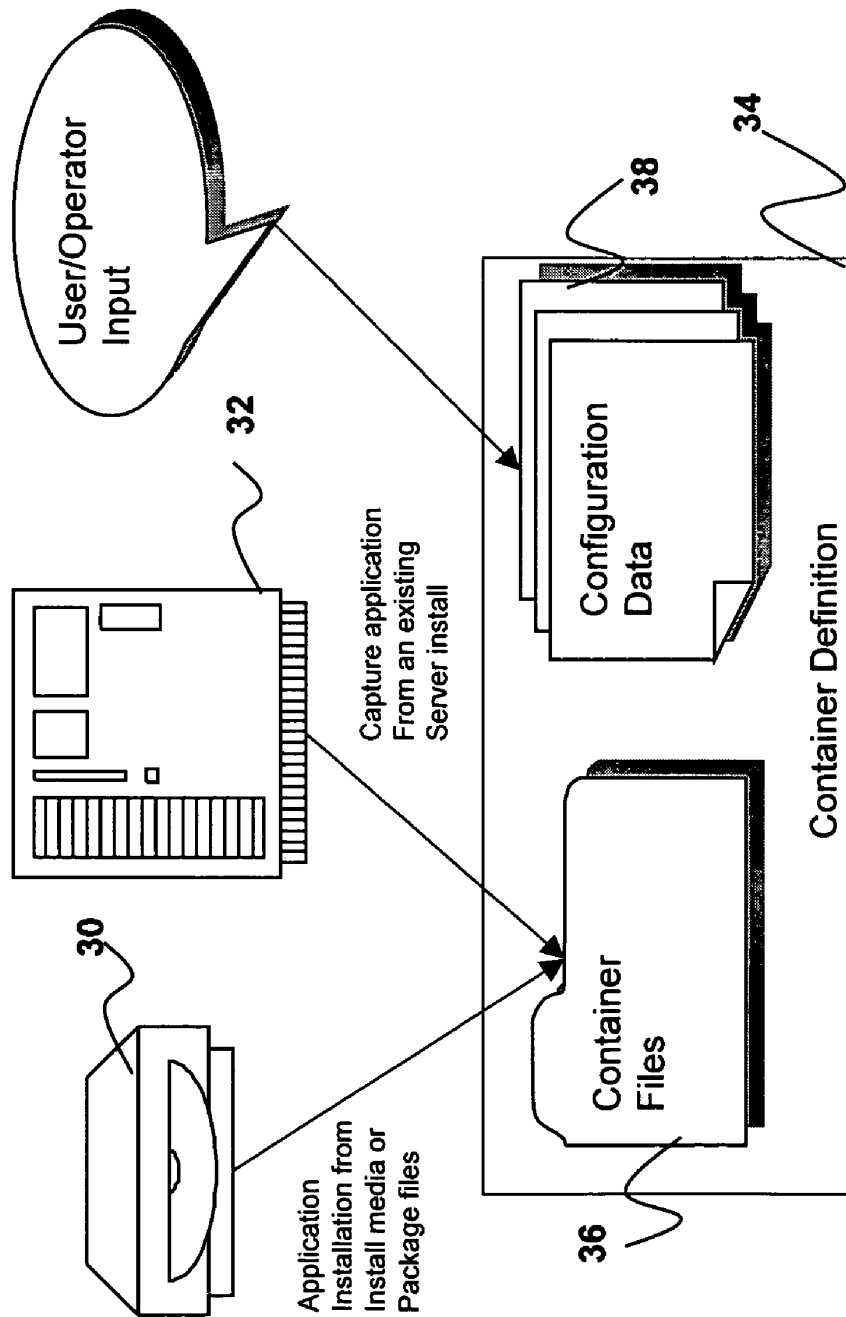


Figure 3

U.S. Patent

Apr. 14, 2009

Sheet 4 of 17

US 7,519,814 B2

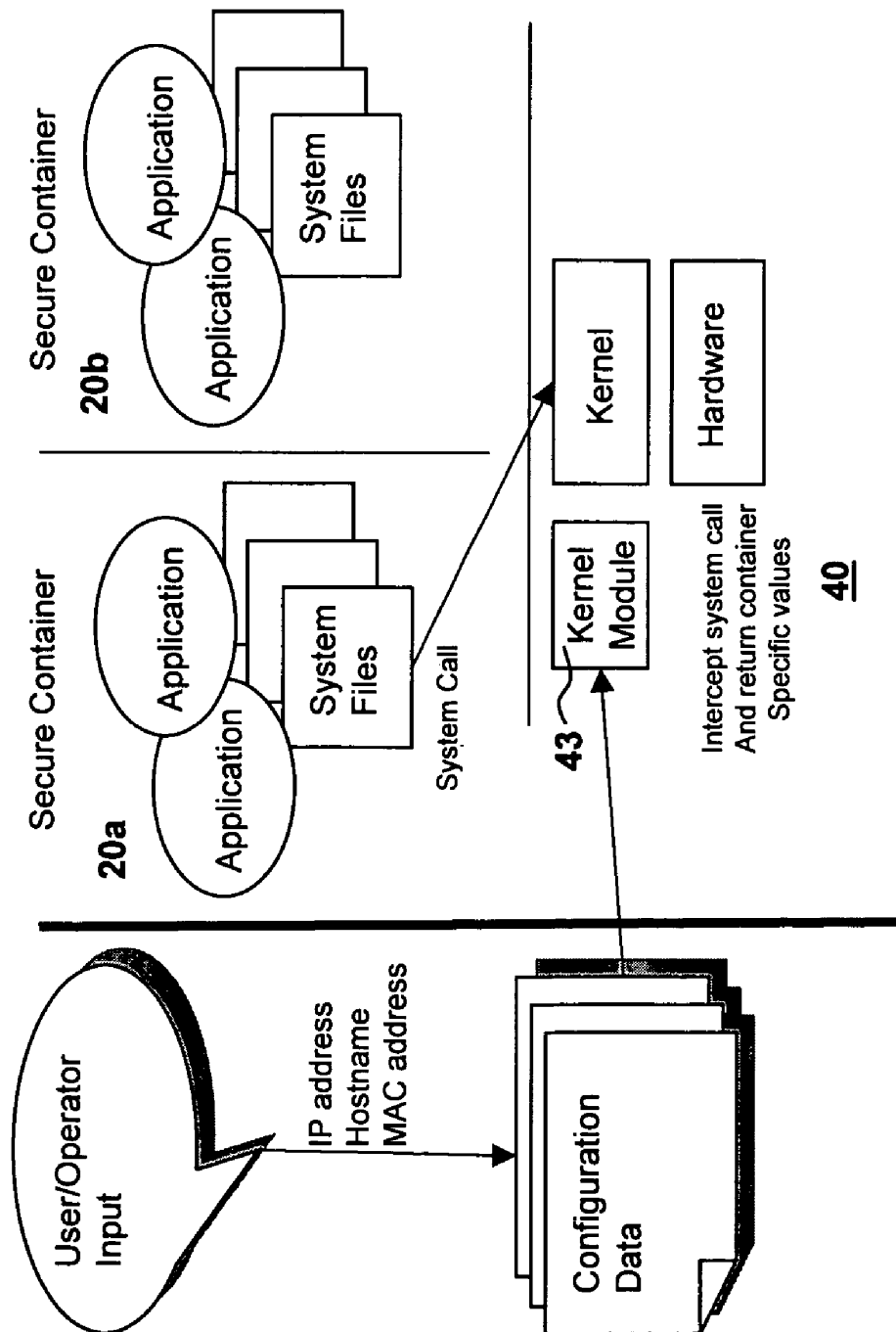


Figure 4

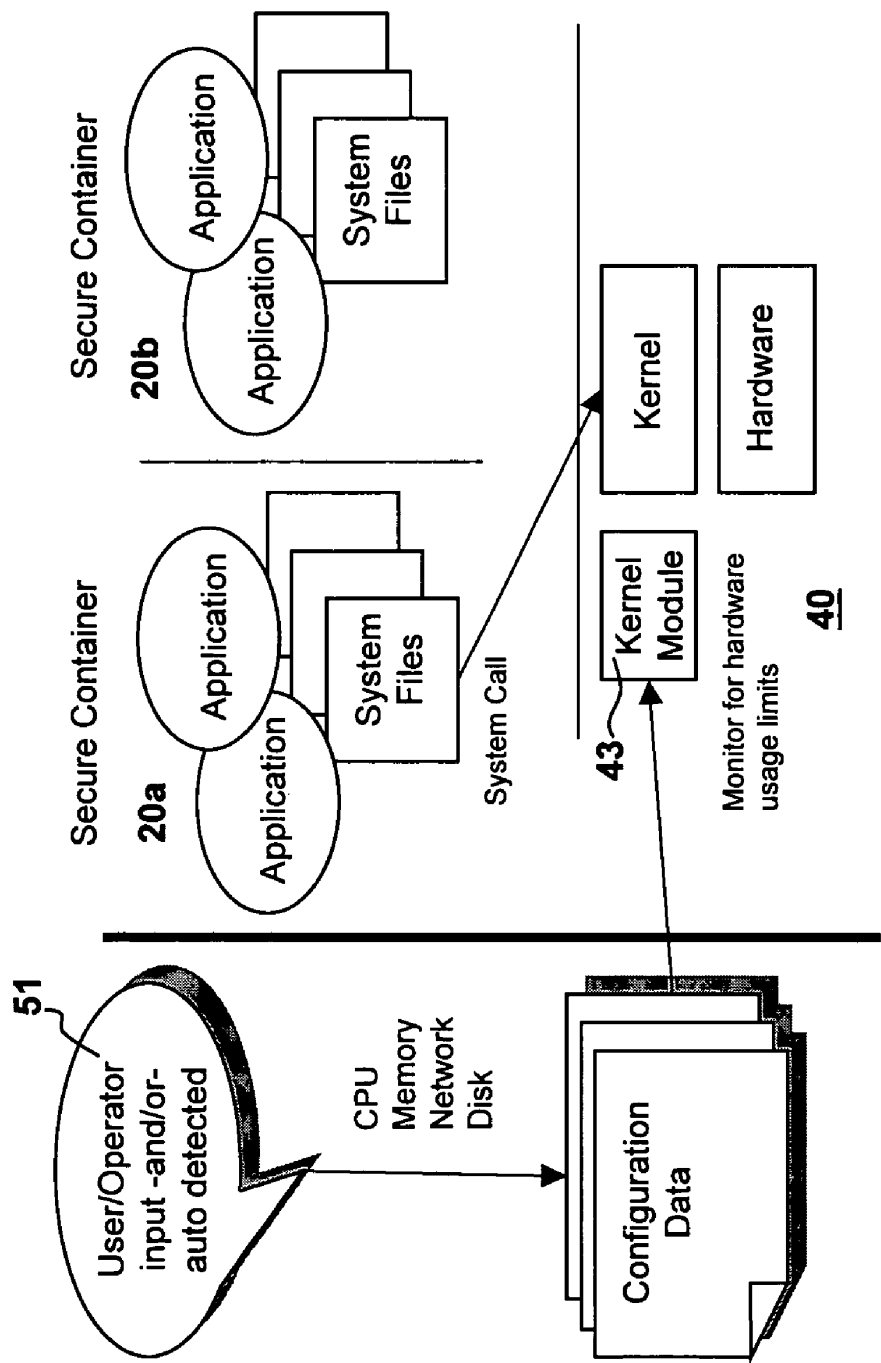
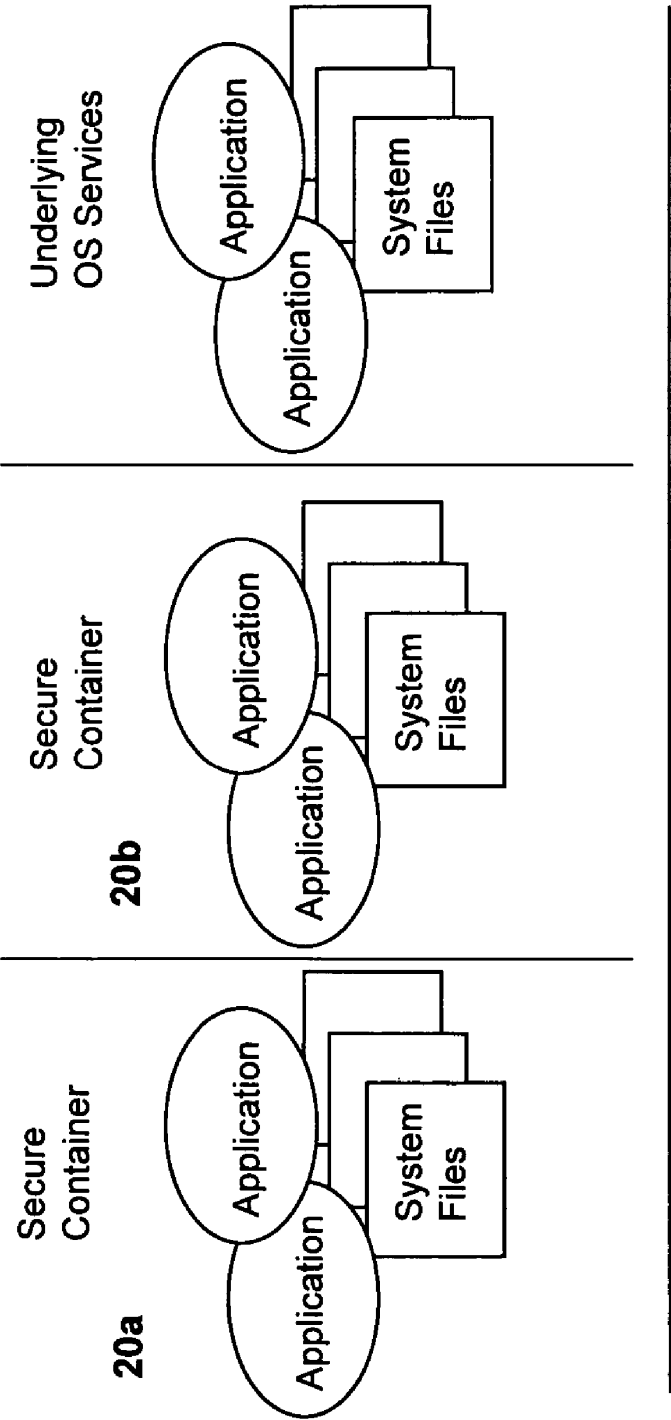


Figure 5



40

Figure 6

U.S. Patent

Apr. 14, 2009

Sheet 7 of 17

US 7,519,814 B2

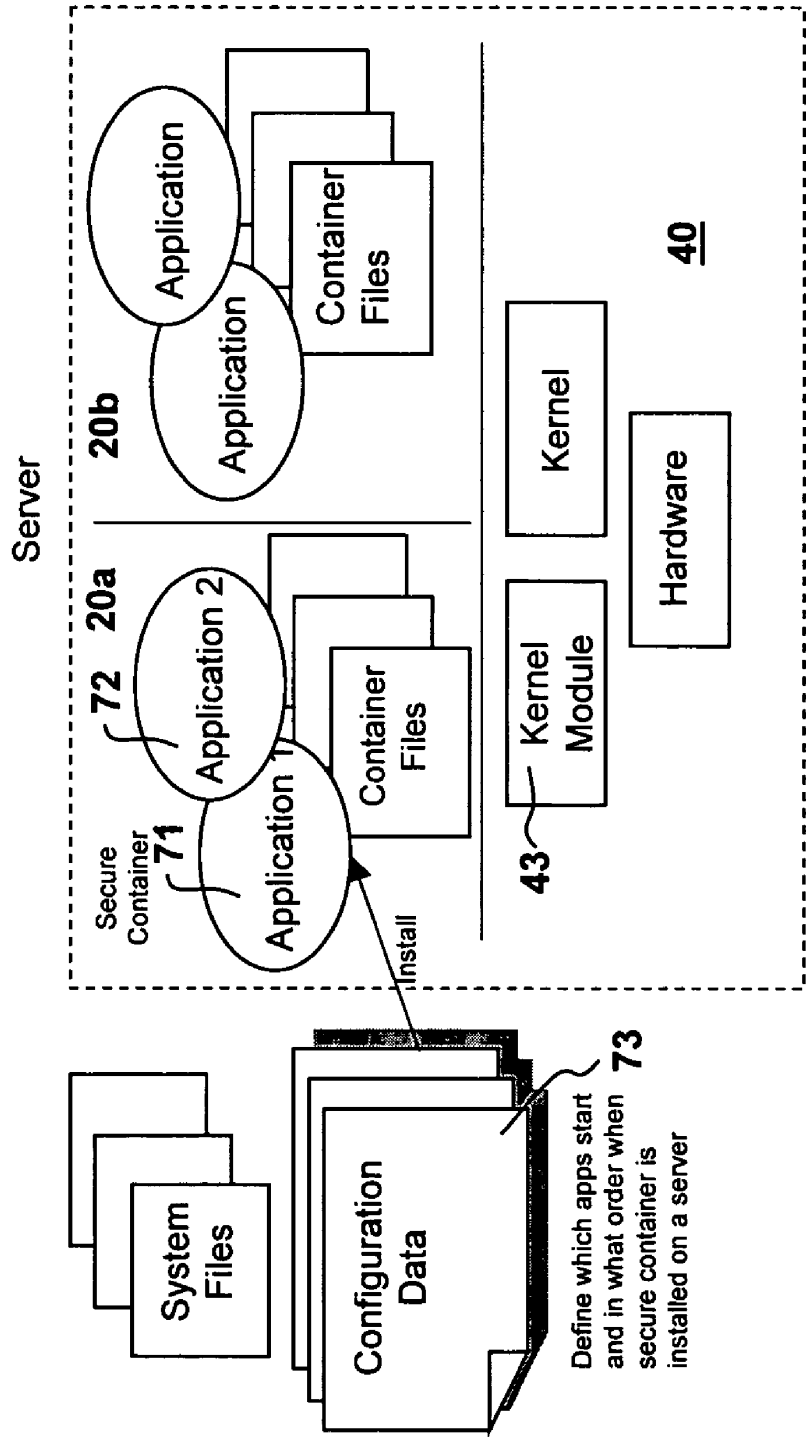


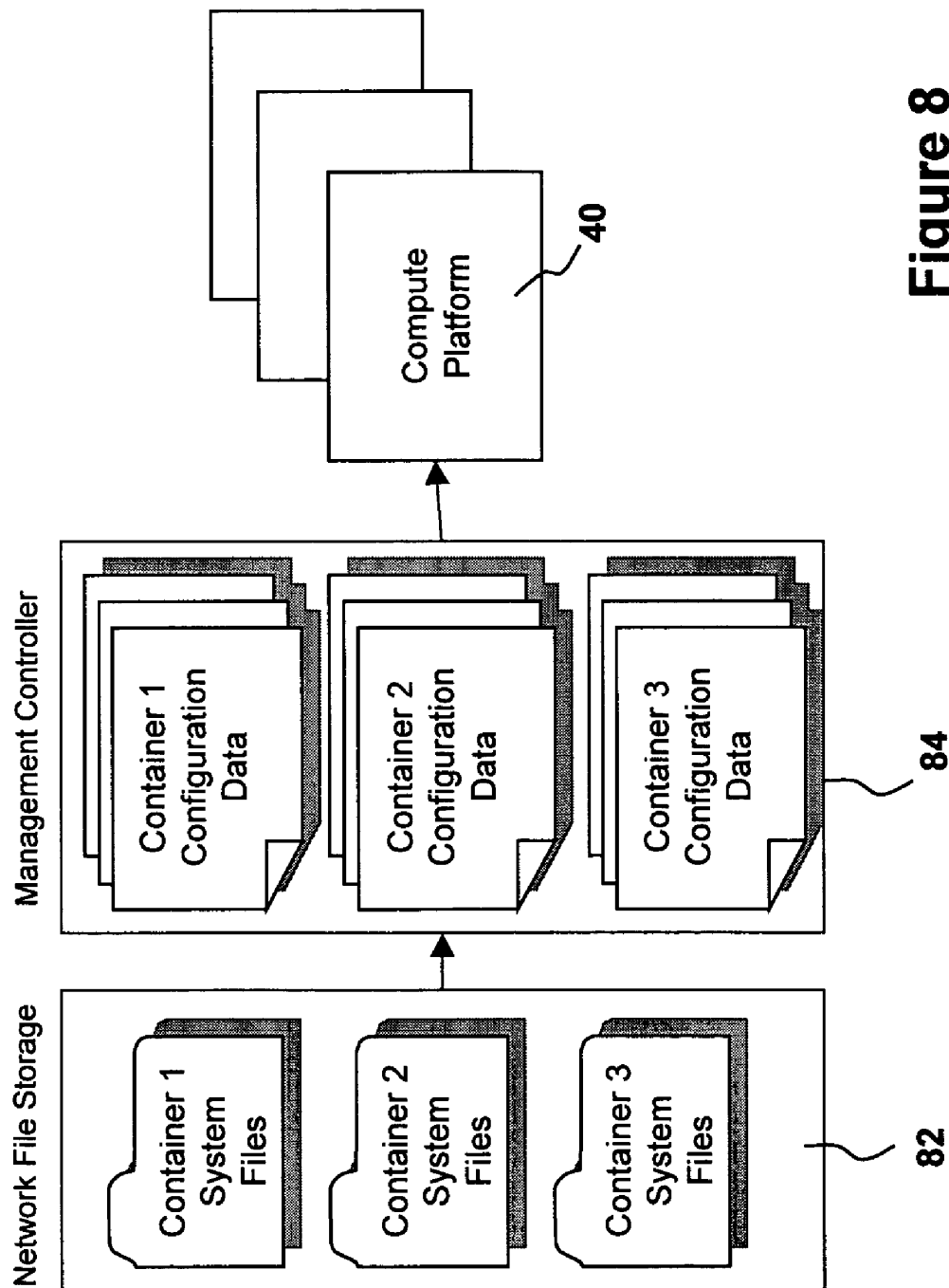
Figure 7

U.S. Patent

Apr. 14, 2009

Sheet 8 of 17

US 7,519,814 B2



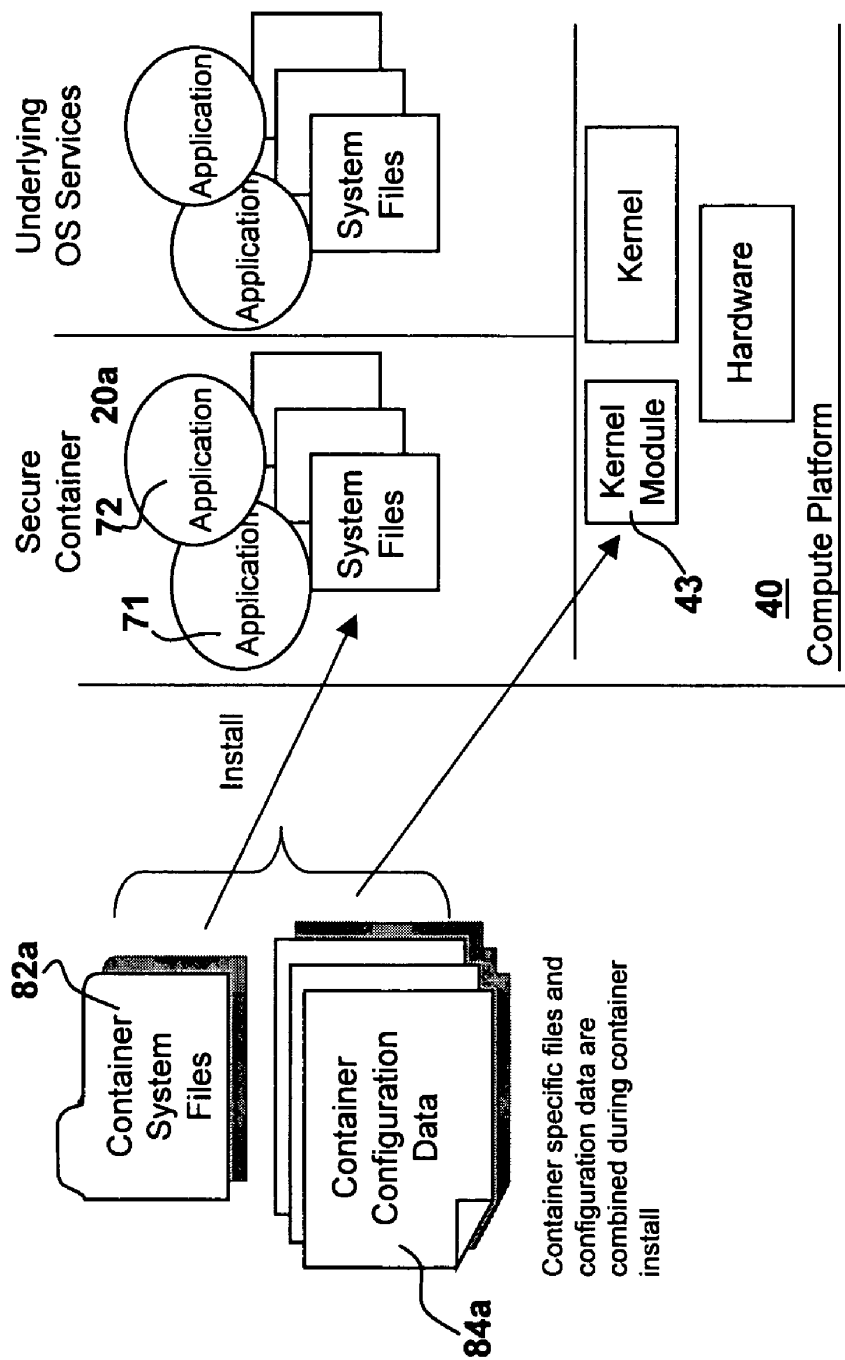


Figure 9

U.S. Patent

Apr. 14, 2009

Sheet 10 of 17

US 7,519,814 B2

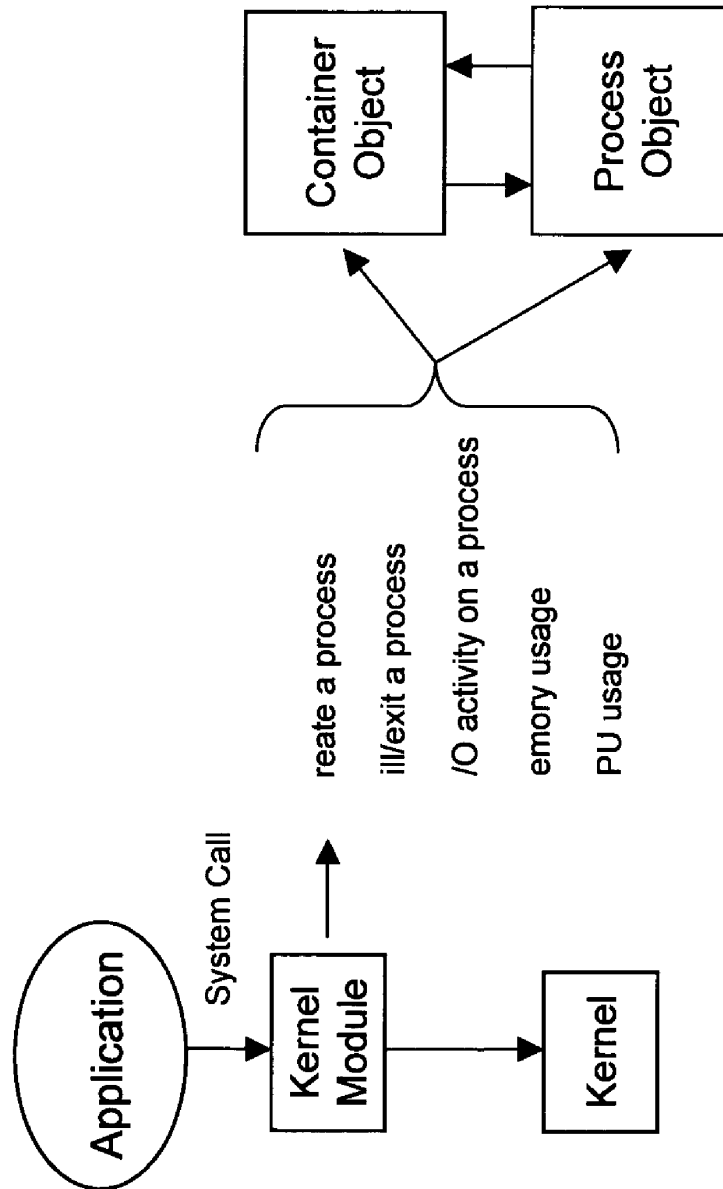


Figure 10

U.S. Patent

Apr. 14, 2009

Sheet 11 of 17

US 7,519,814 B2

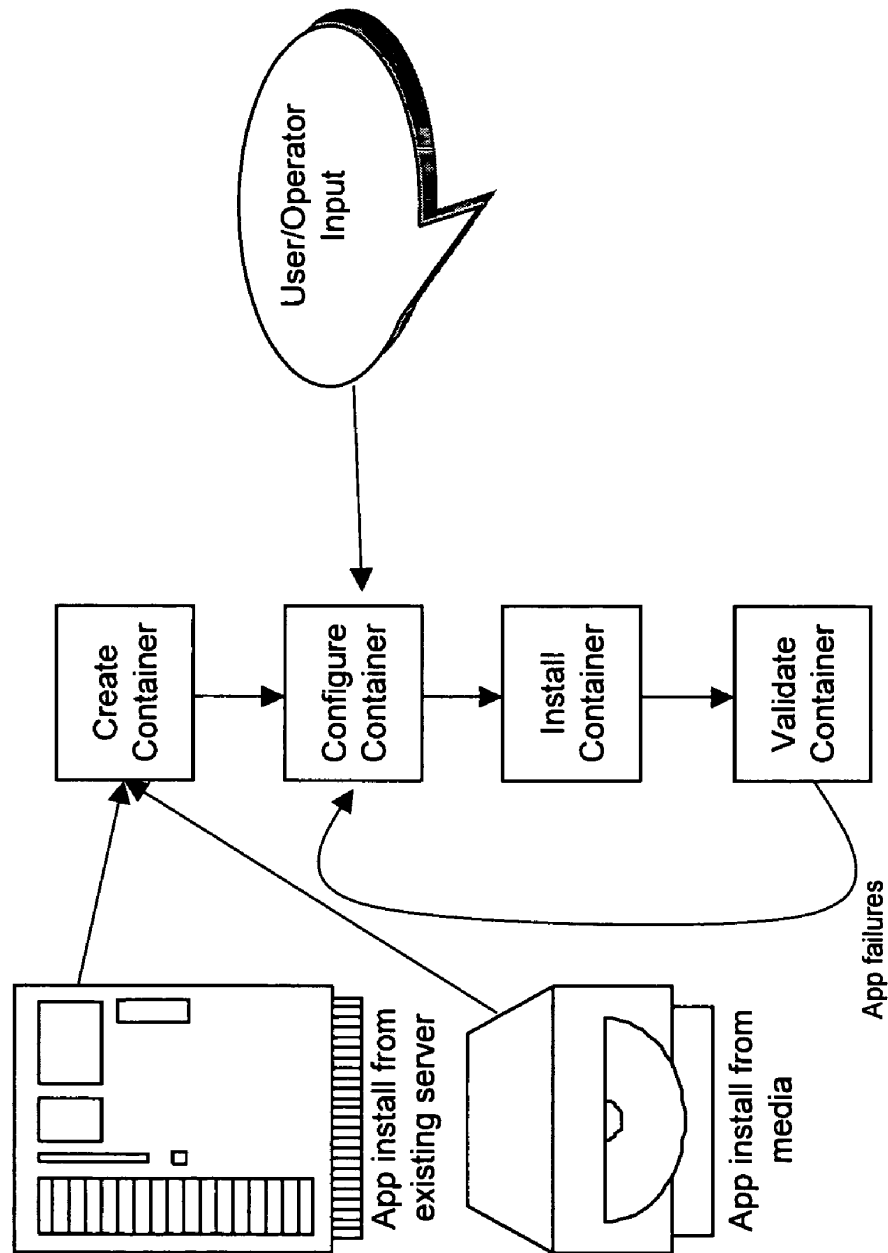


Figure 11

U.S. Patent

Apr. 14, 2009

Sheet 12 of 17

US 7,519,814 B2

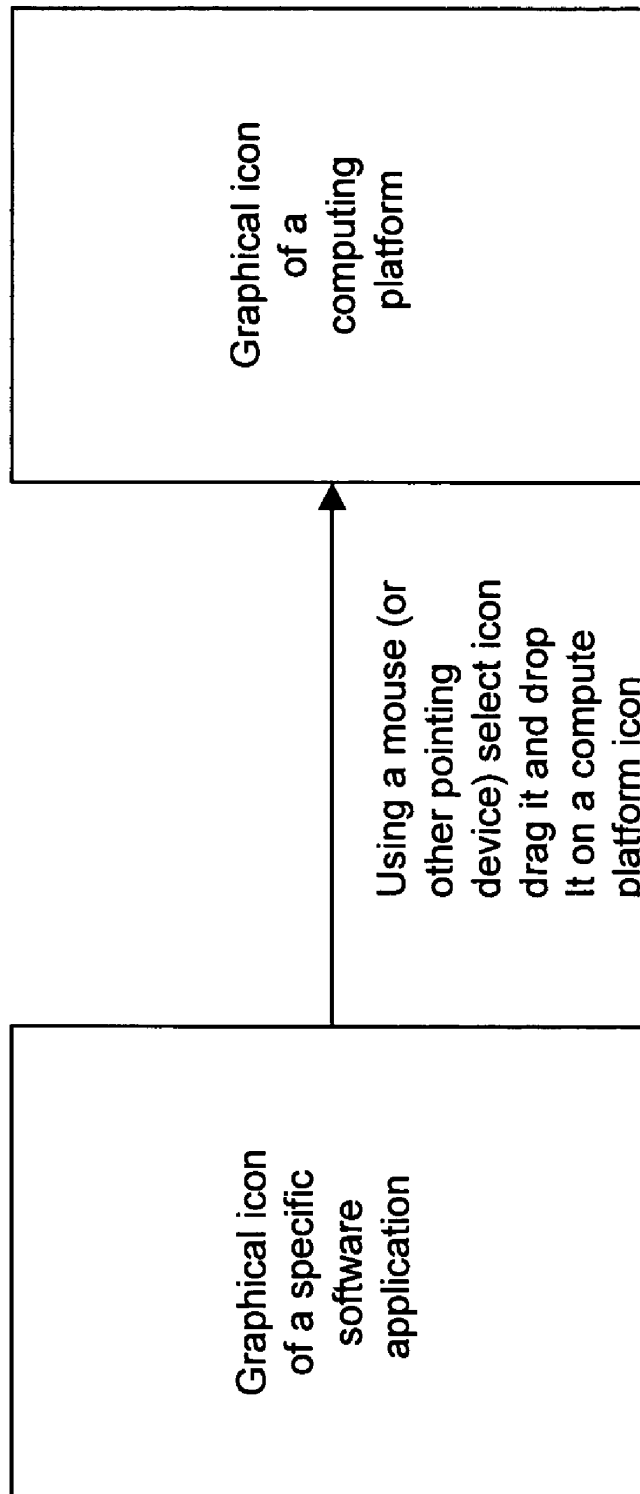


Figure 12

U.S. Patent

Apr. 14, 2009

Sheet 13 of 17

US 7,519,814 B2

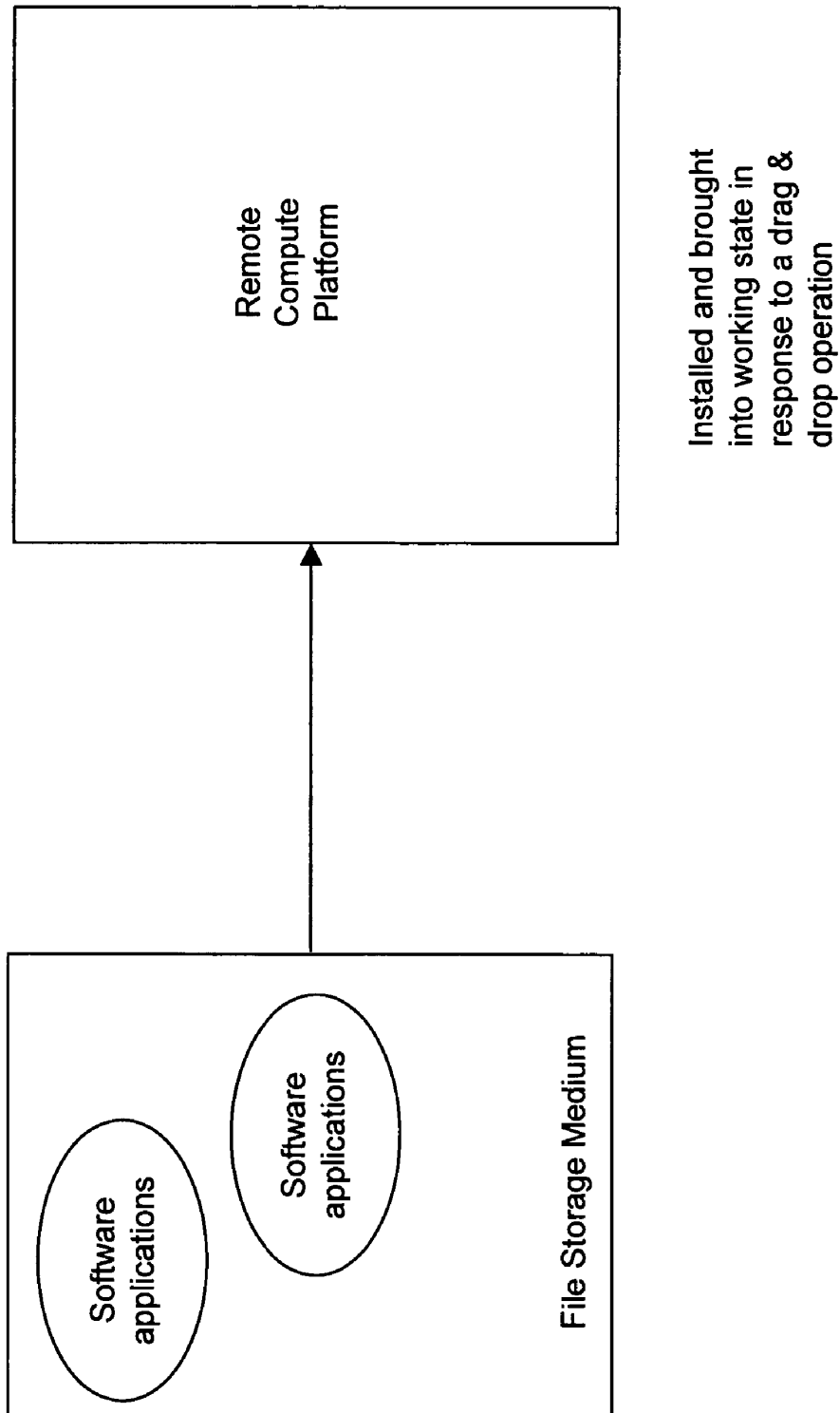


Figure 13

U.S. Patent

Apr. 14, 2009

Sheet 14 of 17

US 7,519,814 B2

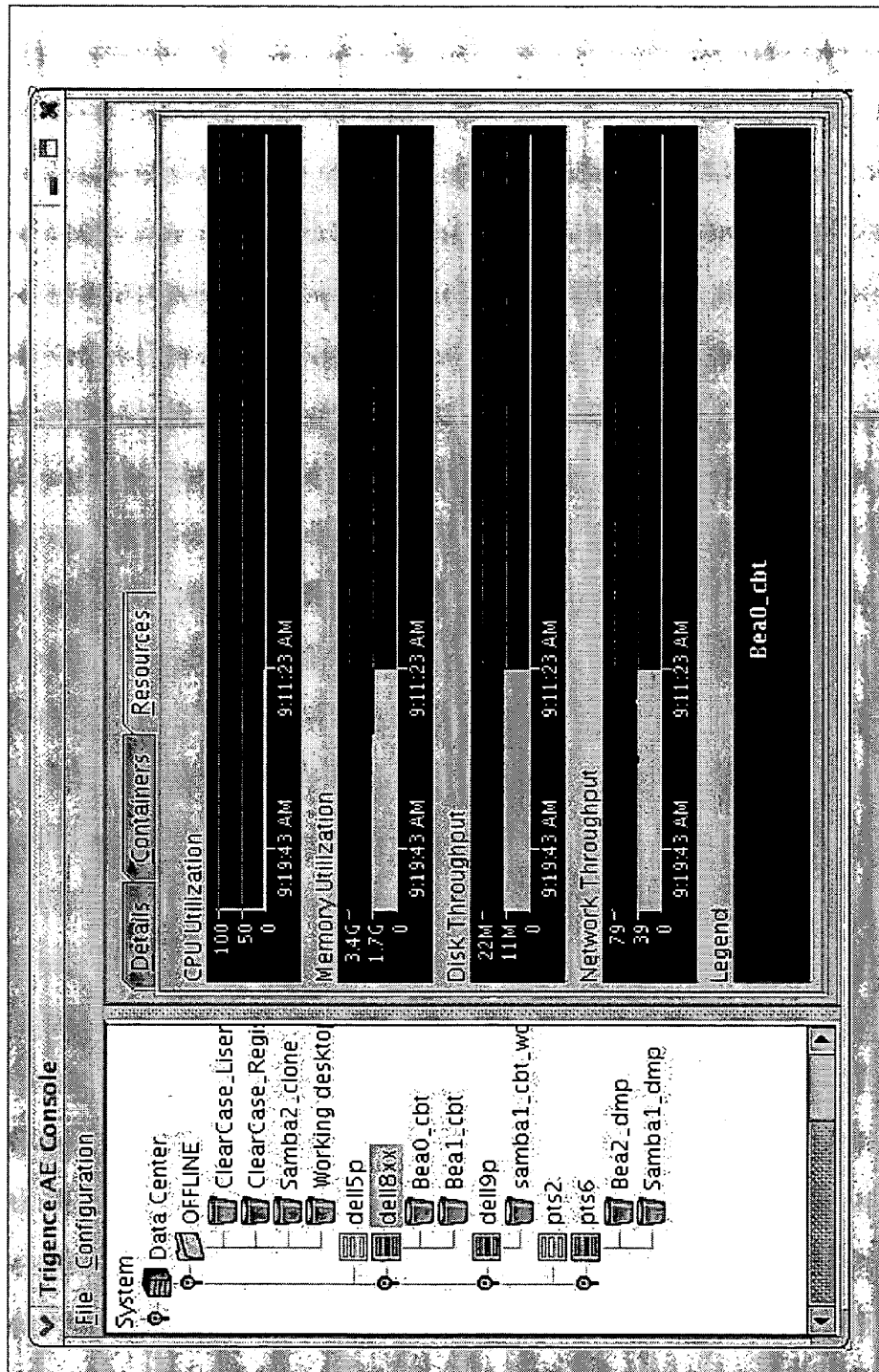


Figure 14

U.S. Patent

Apr. 14, 2009

Sheet 15 of 17

US 7,519,814 B2

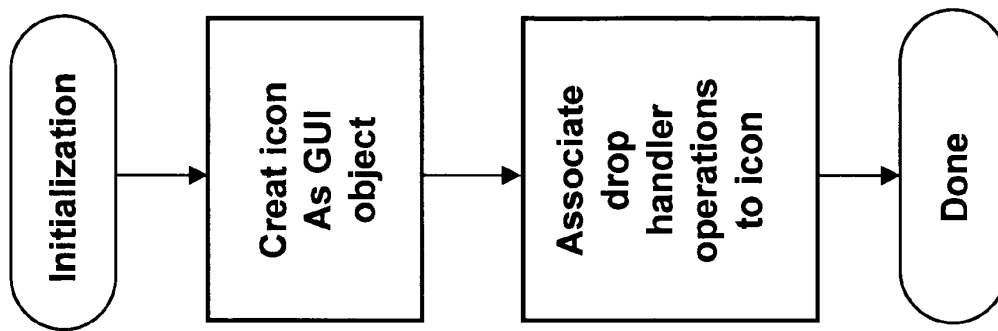


Figure 15

U.S. Patent

Apr. 14, 2009

Sheet 16 of 17

US 7,519,814 B2

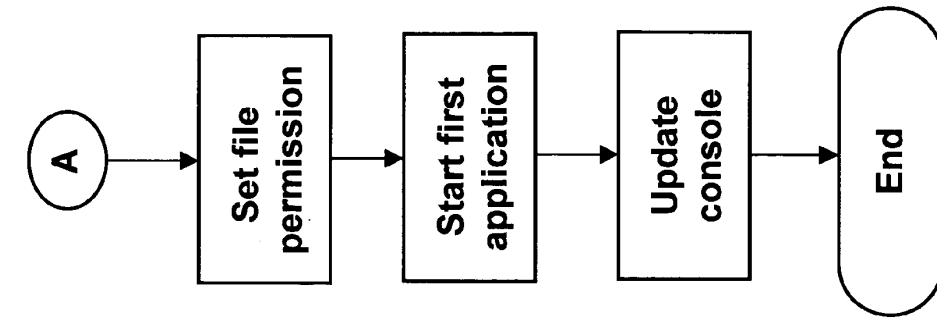
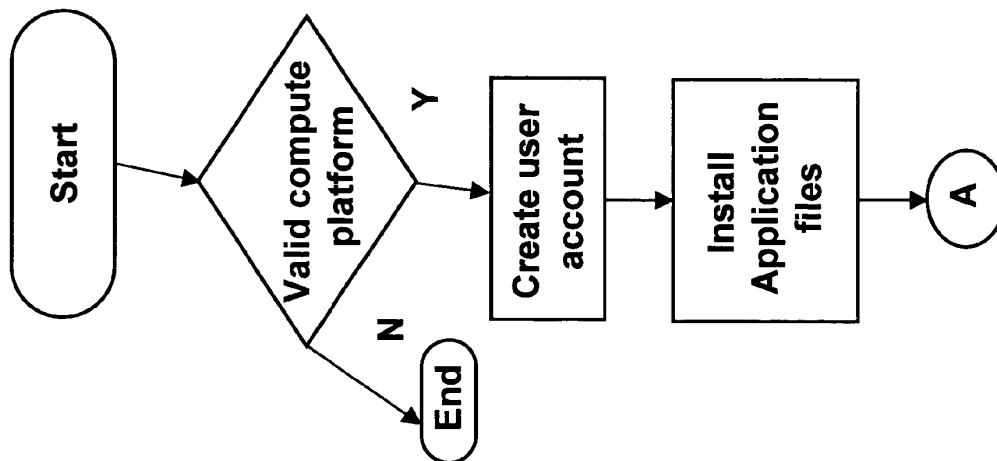


Figure 16



U.S. Patent

Apr. 14, 2009

Sheet 17 of 17

US 7,519,814 B2

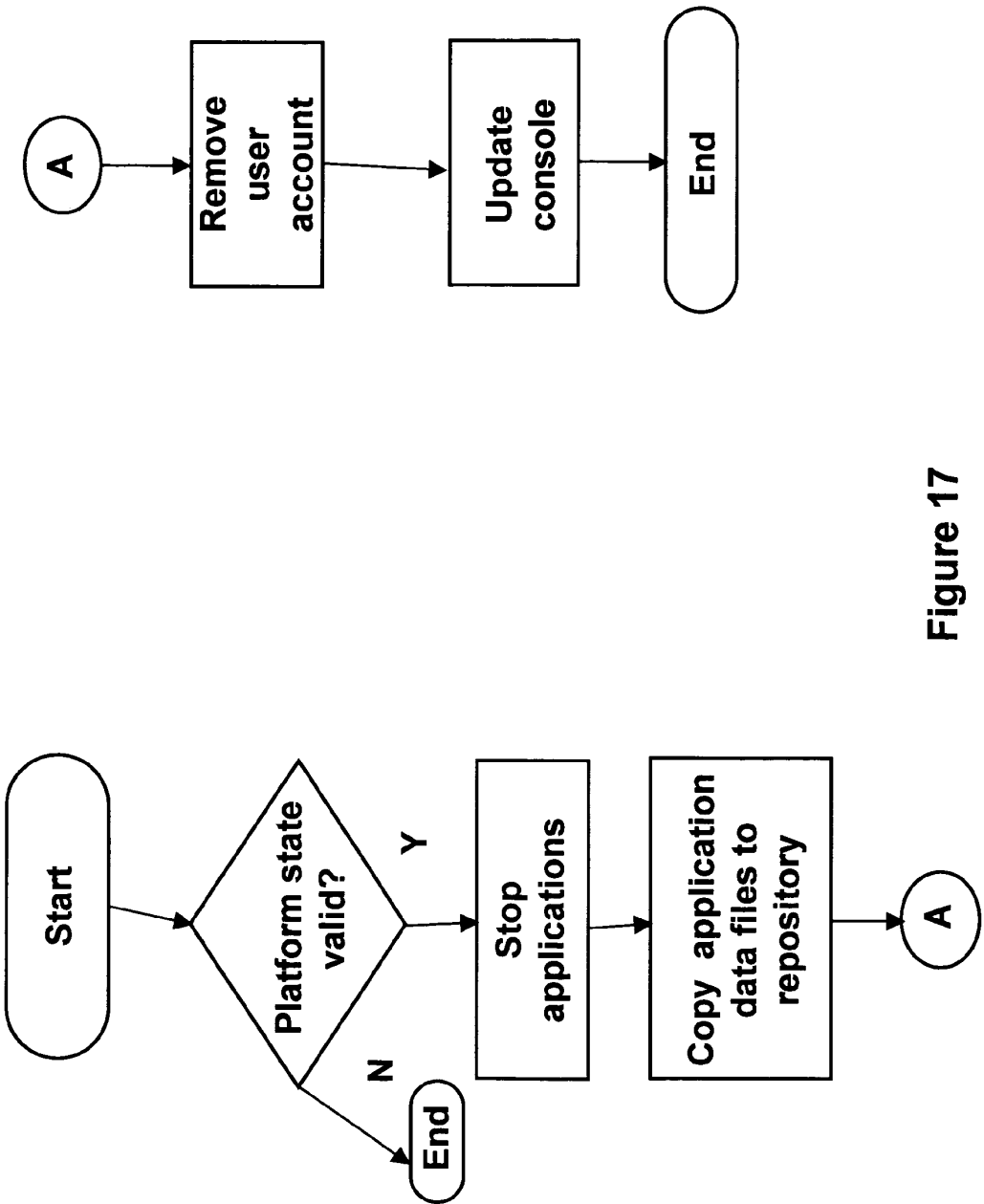


Figure 17

US 7,519,814 B2

1

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**CROSS-REFERENCE TO RELATED APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

2

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

```
/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1
```

US 7,519,814 B2

3

/lib/libnsl.so.1
 /lib/libdl.so.2
 /lib/i686/libpthread.so.0
 /lib/i686/libc.so.6
 /lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
 /etc/httpd/conf
 /etc/httpd/conf.d
 /etc/httpd/logs
 /etc/httpd/modules
 /etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

4

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySql and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

US 7,519,814 B2

5

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of

6

this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed “spoofing” and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. 3 is a pictorial diagram illustrating the creation of a container.

FIG. 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the “kernel module” which is used to handle system calls.

FIG. 5 is a diagram similar to FIG. 4, wherein additional configuration data is provided.

FIG. 6 is a diagram illustrating a system wherein the containers are secure containers.

FIG. 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. 9 is a diagram that illustrates the installation of a container on a server.

FIG. 10 is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. 14 is a screen snapshot of an implementation according to the schematic of FIG. 13.

FIG. 15 is a flow chart of a method of initializing icons of FIG. 14.

FIG. 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. 14; and,

FIG. 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13.

DETAILED DESCRIPTION

Turning now to FIG. 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c.

US 7,519,814 B2

7

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. 2 illustrates a system in accordance with the invention in which multiple applications **21a**, **21b**, **21c**, **21d**, **21e**, and **21f** can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20a**, **20b** and **20c**. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20a**, **20b** and **20c** are shown in FIG. 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. 4 illustrates a system having two secure containers **20a** **20b**, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20a**, **20b** environment system calls are intercepted, by the kernel module **43** passing through services installed as part of

8

this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

As introduced heretofore, “spoofing” is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system’s kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the ‘uname’ system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has ‘spoofed’ the ‘uname’ system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same “spoofing” mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

- tracking applications, for example a tracking of which applications are in and out of a specific container;
- spoofing particular values returned by the kernel;
- applying resource checks and imposing resource limits;
- monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,
- retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20a** or **20b**. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. 6 that each application executing associated with a container **20a** or **20b**, or contained within a

US 7,519,814 B2

9

container **20a** or **20b**, is able to access files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20a** association are not able to access the files contained in another container **20b**.

When a container **20a** or **20b** is installed specific applications are started, as is shown in FIG. 7 and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. 7, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. 8, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. 9 shows that the files **82a**, **84a** are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing

10

the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. 9.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform and then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. 3, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

US 7,519,814 B2

11

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

- modifying the contents a file to add a container specific IP address;
- modifying the contents of a directory to define start scripts that should be executed;
- modifying the contents of a file to define which mount points will relate to a container;
- removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications

12

are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

US 7,519,814 B2

13

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

14

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

US 7,519,814 B2

15

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. 14, a screen snapshot of an implementation is shown according to the schematic of FIG. 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. 15, shown is a flow chart of a method of initializing the icons of FIG. 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. 16 are performed by an install drop handler. During installation, verification is performed to

16

determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. 15), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

US 7,519,814 B2

17

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

18

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

- a) running an instance of a service on a server;
- b) determining which files are being used; and,
- c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim 14 comprising the steps of:

- assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
- installing the container on a server; and,
- testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

- using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with a unique icon representing

US 7,519,814 B2

19

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim 20 further comprising starting a distributed software application.

22. A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim 17 further comprising creating a user account for the service.

25. A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim 1, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;
displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

20

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim 31, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

Exhibit 2

U.S. Patent No. 7,519,814 vs. Microsoft

Accused Instrumentalities: Microsoft products and services using secure containerized applications, including without limitation Azure Kubernetes Service (“AKS”), Azure Arc-enabled Kubernetes, Azure Container Registry, and Azure Container Apps, and all versions and variations thereof since the issuance of the asserted patent.









Claim 1

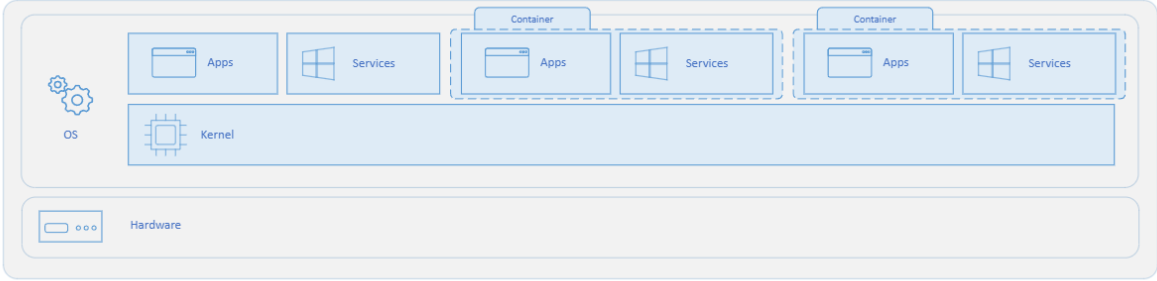
Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>To the extent the preamble is limiting, Microsoft and/or its customer practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p>For example, Azure Kubernetes Service runs on individual servers, each of which runs an independent operating system running either on bare metal, through an on-premises virtualized infrastructure, through one or more cloud services, or through any other supported deployment. In an exemplary deployment, two or more servers use different operating systems. The servers operate in disparate computing environments, including because each server is a stand-alone computer and/or each server is unrelated to the other servers due to having independent hardware and, in some instances, independent software.</p> <p>Microsoft requires and/or provides that each server includes a processor with one or more cores available to the OS kernel. Microsoft further requires and/or provides that each server has a supported operating system, which includes a kernel and associated local system files, configuration files, etc. In the infringing system, at least two servers have different operating systems.</p> <p>In at least some instances, Microsoft directly owns, operates, controls, and/or benefits from the claimed system and/or method. In other instances, Microsoft’s customer makes and uses the system and/or method either by following Microsoft’s direction and control, including Microsoft’s documentation, or automatically through the ordinary and expected operation of Microsoft’s software, or a combination thereof.</p>

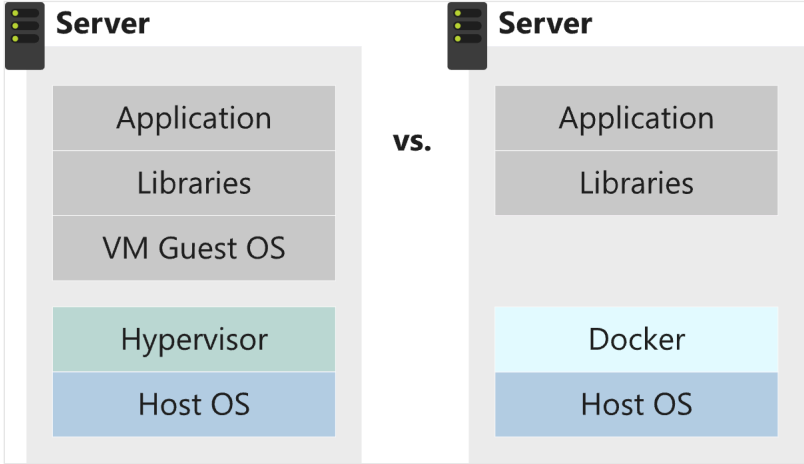
Claim 1	Accused Instrumentalities
	<p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>Azure Kubernetes Service (AKS) is a managed Kubernetes service that you can use to deploy and manage containerized applications. You need minimal container orchestration expertise to use AKS. AKS reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure. AKS is an ideal platform for deploying and managing containerized applications that require high availability, scalability, and portability, and for deploying applications to multiple regions, using open-source tools, and integrating with existing DevOps tools.</p> <p>https://learn.microsoft.com/en-us/azure/aks/what-is-aks</p> <h2>When to use AKS</h2> <p>The following list describes some common use cases for AKS:</p> <ul style="list-style-type: none"> • Lift and shift to containers with AKS: Migrate existing applications to containers and run them in a fully managed Kubernetes environment. • Microservices with AKS: Simplify the deployment and management of microservices-based applications with streamlined horizontal scaling, self-healing, load balancing, and secret management. • Secure DevOps for AKS: Efficiently balance speed and security by implementing secure DevOps with Kubernetes. • Bursting from AKS with ACI: Use virtual nodes to provision pods inside ACI that start in seconds and scale to meet demand. • Machine learning model training with AKS: Train models using large datasets with familiar tools, such as TensorFlow and Kubeflow. • Data streaming with AKS: Ingest and process real-time data streams with millions of data points collected via sensors, and perform fast analyses and computations to develop insights into complex scenarios. • Using Windows containers on AKS: Run Windows Server containers on AKS to modernize your Windows applications and infrastructure. <p>https://learn.microsoft.com/en-us/azure/aks/what-is-aks</p>

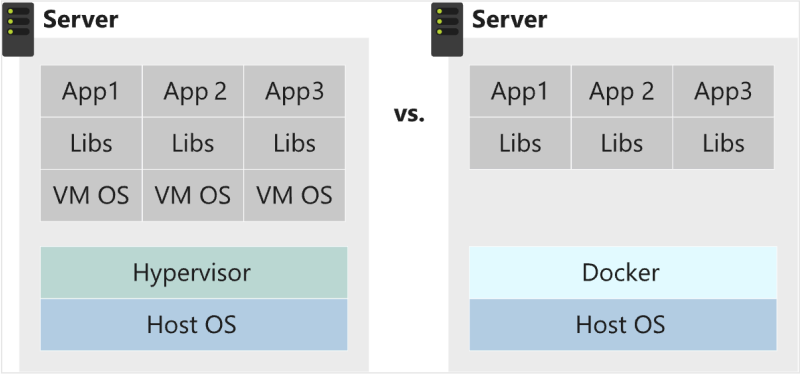
Claim 1	Accused Instrumentalities
	<p>Azure Arc-enabled Kubernetes allows you to attach Kubernetes clusters running anywhere so that you can manage and configure them in Azure. By managing all of your Kubernetes resources in a single control plane, you can enable a more consistent development and operation experience, helping you run cloud-native apps anywhere and on any Kubernetes platform.</p> <p>When the Azure Arc agents are deployed to the cluster, an outbound connection to Azure is initiated, using industry-standard SSL to secure data in transit.</p> <p>Clusters that you connect to Azure are represented as their own resources in Azure Resource Manager, and they can be organized using resource groups and tagging.</p> <p>https://learn.microsoft.com/en-us/azure/azure-arc/kubernetes/overview</p> <p>Containers are becoming the preferred way to package, deploy, and manage cloud applications. Azure Container Instances offers the fastest and simplest way to run Linux or Windows containers in Azure, without having to manage any virtual machines and without having to adopt a higher-level service.</p> <p>ACI supports regular, confidential, and Spot containers. ACI can be used as single-instance or multi-instance via NGroups, or you can get orchestration capabilities by deploying pods in your Azure Kubernetes Service (AKS) cluster via virtual nodes on ACI. For even faster startup times, ACI supports standby pools.</p> <p>https://learn.microsoft.com/en-us/azure/container-instances/container-instances-overview</p>

Claim 1	Accused Instrumentalities
	<p>Azure Container Apps is a serverless platform that allows you to maintain less infrastructure and save costs while running containerized applications. Instead of worrying about server configuration, container orchestration, and deployment details, Container Apps provides all the up-to-date server resources required to keep your applications stable and secure.</p> <p>Common uses of Azure Container Apps include:</p> <ul style="list-style-type: none">• Deploying API endpoints• Hosting background processing jobs• Handling event-driven processing• Running microservices <p>https://learn.microsoft.com/en-us/azure/container-apps/overview</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="737 277 1577 305">Applies to: Windows Server 2022, Windows Server 2019, Windows Server 2016</p> <p data-bbox="737 363 1923 542">Containers are a technology for packaging and running Windows and Linux applications across diverse environments on-premises and in the cloud. Containers provide a lightweight, isolated environment that makes apps easier to develop, deploy, and manage. Containers start and stop quickly, making them ideal for apps that need to rapidly adapt to changing demand. The lightweight nature of containers also make them a useful tool for increasing the density and utilization of your infrastructure.</p> <div data-bbox="737 570 1566 1070"><div data-bbox="737 570 999 1070"><p data-bbox="795 591 940 623">Anywhere</p><p data-bbox="800 802 936 829">On-premises</p><p data-bbox="835 980 898 1008">Cloud</p></div><div data-bbox="1020 570 1283 1070"><p data-bbox="1089 591 1213 623">Any app</p><p data-bbox="1104 802 1199 829">Monolith</p><p data-bbox="1083 980 1213 1008">Microservice</p></div><div data-bbox="1304 570 1566 1070"><p data-bbox="1335 591 1535 623">Any language</p><p data-bbox="1446 704 1493 724">Java</p><p data-bbox="1446 802 1493 821">.Net</p><p data-bbox="1446 899 1524 919">Python</p><p data-bbox="1451 992 1514 1011">Node</p></div></div> <p data-bbox="714 1086 1598 1114">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p>

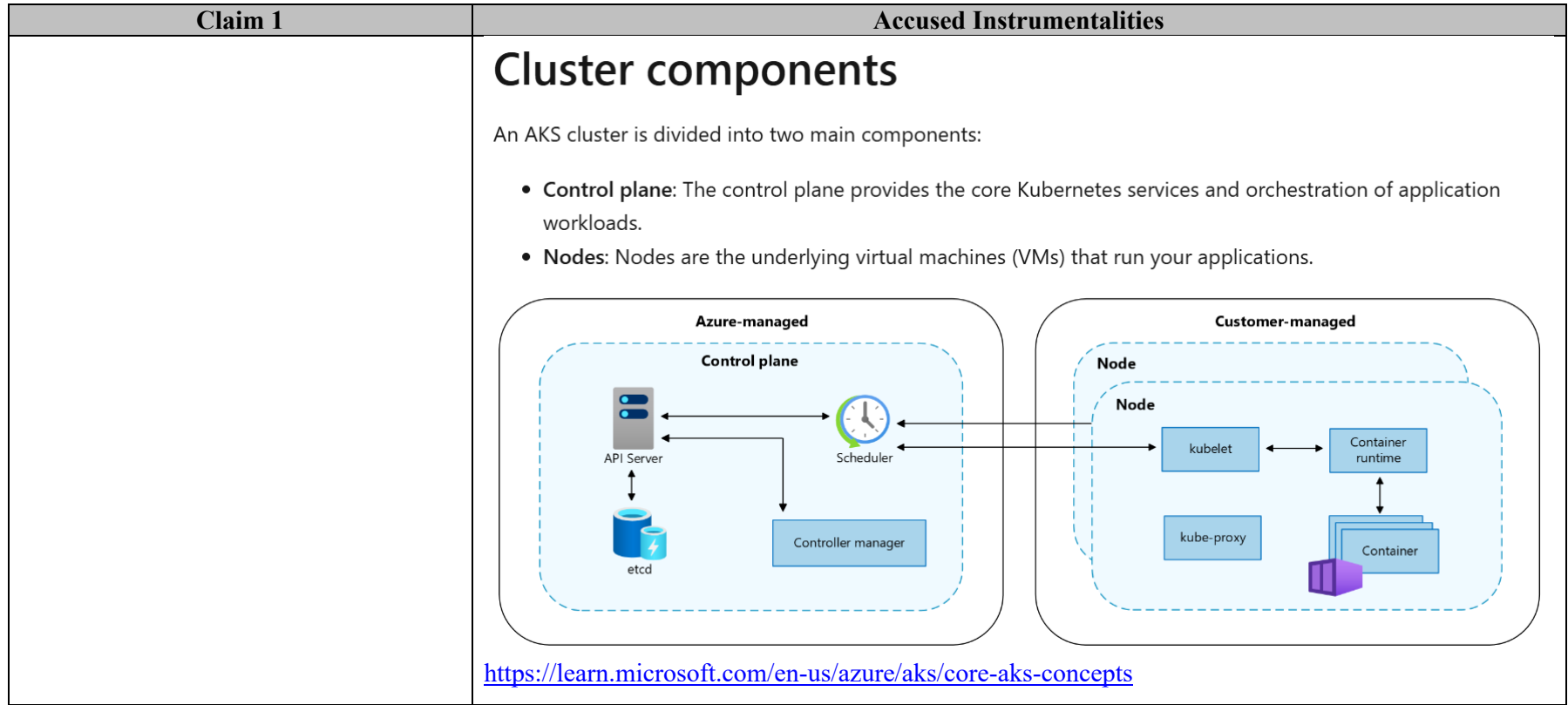
Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 289 1218 337">How containers work</h2> <p data-bbox="730 373 1921 475">A container is an isolated, lightweight package for running an application on the host operating system. Containers build on top of the host operating system's kernel (which can be thought of as the buried plumbing of the operating system), as shown in the diagram below.</p>  <p data-bbox="730 857 1921 1036">While a container shares the host operating system's kernel, the container doesn't get unfettered access to it. Instead, the container gets an isolated—and in some cases virtualized—view of the system. For example, a container can access a virtualized version of the file system and registry, but any changes affect only the container and are discarded when it stops. To save data, the container can mount persistent storage such as an Azure Disk or a file share (including Azure Files).</p> <p data-bbox="730 1073 1921 1287">A container builds on top of the kernel, but the kernel doesn't provide all of the APIs and services an app needs to run—most of these are provided by system files (libraries) that run above the kernel in user mode. Because a container is isolated from the host's user mode environment, the container needs its own copy of these user mode system files, which are packaged into something known as a base image. The base image serves as the foundational layer upon which your container is built, providing it with operating system services not provided by the kernel. But we'll talk more about container images later.</p> <p data-bbox="730 1317 1598 1344">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="722 264 1026 305"><h2>Docker benefits</h2></div> <div data-bbox="722 337 1530 362"><p>When we use Docker, we immediately get access to the benefits containerization offer.</p></div> <div data-bbox="722 410 1073 446"><h3>Efficient hardware use</h3></div> <div data-bbox="722 482 1913 539"><p>Containers run without using a virtual machine (VM). As we learned, the container relies on the host kernel for functions such as file system, network management, process scheduling, and memory management.</p></div> <div data-bbox="722 563 1520 1024"><p>The diagram illustrates the architectural differences between a Virtual Machine (VM) and Docker containers. On the left, labeled 'Server', the stack consists of a 'Host OS' at the bottom, followed by a 'Hypervisor', then a 'VM Guest OS', and finally the 'Application' and 'Libraries' layers at the top. On the right, also labeled 'Server', the stack is simpler: 'Host OS' at the bottom, followed by 'Docker', and then the 'Application' and 'Libraries' layers. A 'vs.' label is placed between the two stacks to indicate a comparison.</p></div> <div data-bbox="722 1060 1913 1153"><p>Compared to a VM, we can see that a VM requires an OS installed to provide kernel functions to the running applications inside the VM. Keep in mind that the VM OS also requires disk space, memory, and CPU time. By removing the VM and the additional OS requirement, we can free resources on the host and use it for running other containers.</p></div> <div data-bbox="714 1175 1890 1239"><p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p></div>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 261 1031 297">Container isolation</p> <p data-bbox="730 329 1915 423">Docker containers provide security features to run multiple containers simultaneously on the same host without affecting each other. As we learned, we can configure both data storage and network configuration to isolate our containers or share data and connectivity between specific containers.</p> <p data-bbox="730 451 1100 477">Let's compare this feature to using VMs.</p> <div data-bbox="730 500 1524 873">  </div> <p data-bbox="730 906 1915 1000">Assume we have a physical host running two VMs. We have three applications that we want to run isolated from each other. We decide to deploy the first app onto VM1 and the second onto VM2 to separate the two apps from each other. If we now choose to install the third application, we'll need to install another VM to continue this pattern.</p> <p data-bbox="714 1016 1890 1081">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="730 1109 1083 1149">Application portability</p> <p data-bbox="730 1182 1915 1240">Containers run almost everywhere: desktops, physical servers, VMs, and in the cloud. This runtime compatibility makes it easy to move containerized applications among different environments.</p> <p data-bbox="714 1256 1890 1321">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 269 1035 310">Cloud deployments</p> <p data-bbox="730 337 1919 399">Docker containers are the default container architecture the Azure containerization services use, and many other cloud platforms also support them.</p> <p data-bbox="730 427 1919 488">For instance, you can deploy Docker containers to Azure Container Instances, Azure App Service, and Azure Kubernetes Services. Each of these options provides you with different features and capabilities.</p> <p data-bbox="730 516 1919 610">For example, Azure container instances allow you to focus on designing and building your applications without the overhead of managing infrastructure. When you have many containers to orchestrate, Azure Kubernetes service makes it easy to deploy and manage large-scale container deployments.</p> <p data-bbox="714 630 1892 696">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="730 735 1919 802">There are many different orchestrators that you can use with Windows containers; here are the options Microsoft provides:</p> <ul data-bbox="766 834 1787 901" style="list-style-type: none">• Azure Kubernetes Service (AKS) - use a managed Azure Kubernetes service• Azure Kubernetes Service (AKS) on Azure Stack HCI - use Azure Kubernetes Service on-premises <p data-bbox="714 948 1598 980">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p>

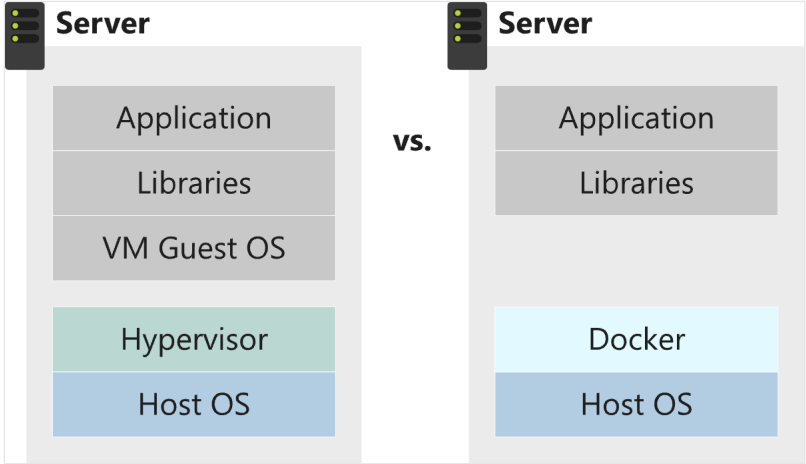
Claim 1	Accused Instrumentalities
	<p>What is Kubernetes?</p> <p>Kubernetes is an open-source container orchestration platform for automating the deployment, scaling, and management of containerized applications. For more information, see the official Kubernetes documentation .</p> <p>What is AKS?</p> <p>AKS is a managed Kubernetes service that simplifies deploying, managing, and scaling containerized applications using Kubernetes. For more information, see What is Azure Kubernetes Service (AKS)?</p> <p>https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts</p>



Claim 1	Accused Instrumentalities
	<h2 data-bbox="726 266 1087 311">VM size and image</h2> <p data-bbox="726 347 1871 490">The Azure VM size for your nodes defines CPUs, memory, size, and the storage type available, such as high-performance SSD or regular HDD. The VM size you choose depends on the workload requirements and the number of pods you plan to run on each node. For more information, see Supported VM sizes in Azure Kubernetes Service (AKS).</p> <p data-bbox="726 526 1919 669">In AKS, the VM image for your cluster's nodes is based on Ubuntu Linux, Azure Linux, or Windows Server 2022. When you create an AKS cluster or scale out the number of nodes, the Azure platform automatically creates and configures the requested number of VMs. Agent nodes are billed as standard VMs, so any VM size discounts, including Azure reservations, are automatically applied.</p> <p data-bbox="714 704 1453 734">https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts</p> <p data-bbox="726 782 1892 902">In order to run Windows containers, your Kubernetes cluster must include multiple operating systems. While you can only run the <u>control plane</u> on Linux, you can deploy worker nodes running either Windows or Linux.</p> <p data-bbox="726 945 1892 1019"><u>Windows nodes</u> are supported provided that the operating system is Windows Server 2019 or Windows Server 2022.</p> <p data-bbox="726 1062 1864 1182">This document uses the term <i>Windows containers</i> to mean Windows containers with process isolation. Kubernetes does not support running Windows containers with Hyper-V isolation.</p> <p data-bbox="714 1224 1320 1253">https://kubernetes.io/docs/concepts/windows/intro/</p>

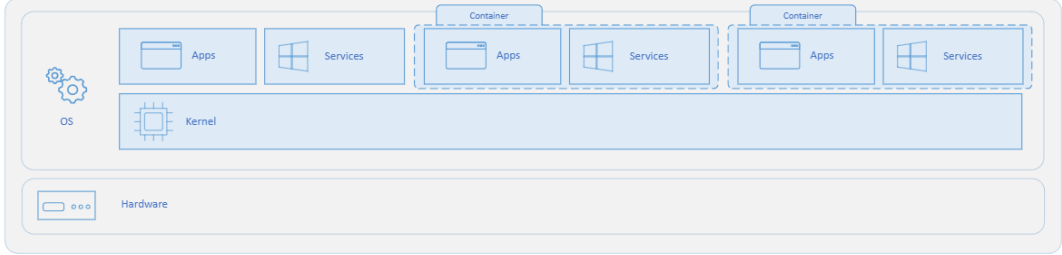
Claim 1	Accused Instrumentalities
	<p data-bbox="728 264 1902 378">Are there any limitations on the number of services on a cluster with Windows nodes?</p> <p data-bbox="728 415 1877 480">A cluster with Windows nodes can have approximately 500 services (sometimes less) before it encounters port exhaustion. This limitation applies to a Kubernetes Service with External Traffic Policy set to "Cluster".</p> <p data-bbox="728 518 1902 656">When the external traffic policy on a Service is configured as a Cluster, the traffic undergoes an extra Source NAT on the node. This process also results in reservation of a port from the TCPIP dynamic port pool. This port pool is a limited resource (~16K ports by default) and many active connections to a Service can lead to dynamic port pool exhaustion resulting in connection drops.</p> <p data-bbox="728 693 1896 758">If the Kubernetes Service is configured with External Traffic Policy set to "Local", port exhaustion problems aren't likely to occur at 500 services.</p> <p data-bbox="714 771 1394 800">https://learn.microsoft.com/en-us/azure/aks/windows-faq</p> <p data-bbox="722 846 1854 1024">Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p data-bbox="714 1057 1457 1086">https://www.techtarget.com/searchdatacenter/definition/kernel</p>

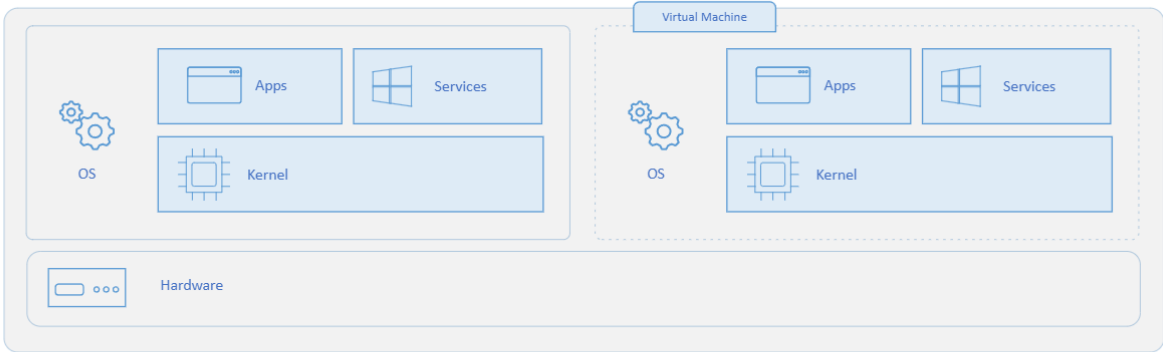
Claim 1	Accused Instrumentalities
<p>[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>The method practiced by Microsoft and/or its customer through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p> <p>For example, OCI and/or OKE stores application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent storage available to each node running the application. The terms “node” and “host” are both used to refer to the claimed server. The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network. In addition to the application software, each container includes associated system files, including a Linux user space required to execute the application, for example including runtime linked libraries (<i>e.g.</i> .dll/.so files), configuration files, Windows services, etc. necessary for the application. For example, the container includes a base OS image provided by Microsoft or by a third party. The container is for use with the host kernel, for example because the application(s) and container libraries are linked against the Linux kernel, and the supported host operating systems also use the Linux kernel, which has a stable binary interface. In another example, the container is for use with the host kernel because the application(s) and Windows services within the container are compatible with the host Windows operating system and kernel.</p> <p>The containers are secure containers as claimed. For example, the data within an individual container is insulated from the effects of other containers except to the extent the container is specifically configured to allow other containers to modify its data, for example using a shared volume.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<div data-bbox="722 264 1029 305"><h2>Docker benefits</h2></div> <div data-bbox="722 337 1533 362"><p>When we use Docker, we immediately get access to the benefits containerization offer.</p></div> <div data-bbox="722 410 1075 446"><h3>Efficient hardware use</h3></div> <div data-bbox="722 482 1915 539"><p>Containers run without using a virtual machine (VM). As we learned, the container relies on the host kernel for functions such as file system, network management, process scheduling, and memory management.</p></div> <div data-bbox="722 563 1522 1024"><p>The diagram illustrates the architectural differences between a Virtual Machine (VM) and Docker containers. On the left, labeled 'Server', the stack consists of five layers: 'Application' (grey), 'Libraries' (grey), 'VM Guest OS' (grey), 'Hypervisor' (teal), and 'Host OS' (blue). On the right, also labeled 'Server', the stack consists of three layers: 'Application' (grey), 'Libraries' (grey), and 'Docker' (light blue), which sits directly on top of the 'Host OS' (blue). A 'vs.' label is placed between the two stacks.</p></div> <div data-bbox="722 1060 1915 1153"><p>Compared to a VM, we can see that a VM requires an OS installed to provide kernel functions to the running applications inside the VM. Keep in mind that the VM OS also requires disk space, memory, and CPU time. By removing the VM and the additional OS requirement, we can free resources on the host and use it for running other containers.</p></div> <div data-bbox="714 1175 1890 1239"><p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p></div>

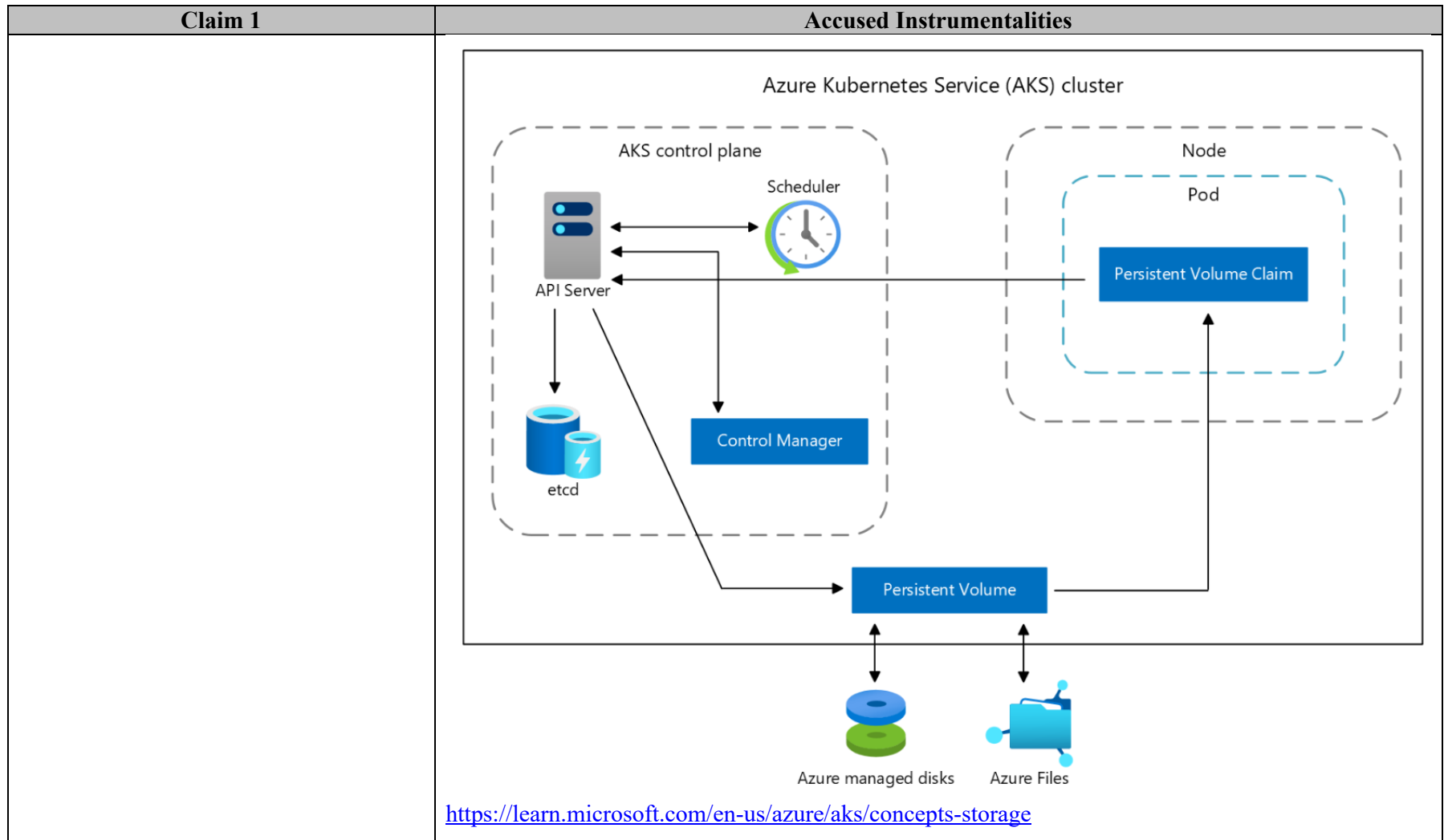
Claim 1	Accused Instrumentalities
	<p data-bbox="730 261 1031 293">Container isolation</p> <p data-bbox="730 329 1915 422">Docker containers provide security features to run multiple containers simultaneously on the same host without affecting each other. As we learned, we can configure both data storage and network configuration to isolate our containers or share data and connectivity between specific containers.</p> <p data-bbox="730 451 1102 475">Let's compare this feature to using VMs.</p> <div data-bbox="730 500 1524 873"> <p>The diagram illustrates two architectural models for running applications on a server. On the left, a 'Server' box contains three separate application stacks, each consisting of an application (App1, App2, App3), its libraries (Libs), and a full VM OS. These VM OSs are managed by a Hypervisor, which sits on top of the Host OS. On the right, a similar 'Server' box shows the same three applications and libraries, but they are managed by Docker containers, which sit directly on top of the Host OS. A 'vs.' label is placed between the two diagrams to highlight the difference in layering and isolation.</p> </div> <p data-bbox="730 906 1915 998">Assume we have a physical host running two VMs. We have three applications that we want to run isolated from each other. We decide to deploy the first app onto VM1 and the second onto VM2 to separate the two apps from each other. If we now choose to install the third application, we'll need to install another VM to continue this pattern.</p> <p data-bbox="714 1015 1890 1079">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="730 1109 1083 1149">Application portability</p> <p data-bbox="730 1182 1915 1239">Containers run almost everywhere: desktops, physical servers, VMs, and in the cloud. This runtime compatibility makes it easy to move containerized applications among different environments.</p> <p data-bbox="714 1255 1890 1320">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 269 1035 310">Cloud deployments</p> <p data-bbox="730 337 1919 399">Docker containers are the default container architecture the Azure containerization services use, and many other cloud platforms also support them.</p> <p data-bbox="730 427 1919 488">For instance, you can deploy Docker containers to Azure Container Instances, Azure App Service, and Azure Kubernetes Services. Each of these options provides you with different features and capabilities.</p> <p data-bbox="730 516 1919 610">For example, Azure container instances allow you to focus on designing and building your applications without the overhead of managing infrastructure. When you have many containers to orchestrate, Azure Kubernetes service makes it easy to deploy and manage large-scale container deployments.</p> <p data-bbox="714 630 1892 691">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <ul data-bbox="730 727 1919 992" style="list-style-type: none">• Deploy containers at scale on Azure or other clouds:<ul style="list-style-type: none">◦ Pull your app (container image) from a container registry, such as the Azure Container Registry, and then deploy and manage it at scale using an orchestrator such as Azure Kubernetes Service (AKS).◦ Azure Kubernetes Service deploys containers to Azure virtual machines and manages them at scale, whether that's dozens of containers, hundreds, or even thousands. The Azure virtual machines run either a customized Windows Server image (if you're deploying a Windows-based app), or a customized Ubuntu Linux image (if you're deploying a Linux-based app). <p data-bbox="714 1011 1598 1040">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 282 1176 329">How containers work</h2> <p data-bbox="730 362 1818 456">A container is an isolated, lightweight package for running an application on the host operating system. Containers build on top of the host operating system's kernel (which can be thought of as the buried plumbing of the operating system), as shown in the diagram below.</p>  <p data-bbox="730 808 1818 971">While a container shares the host operating system's kernel, the container doesn't get unfettered access to it. Instead, the container gets an isolated—and in some cases virtualized—view of the system. For example, a container can access a virtualized version of the file system and registry, but any changes affect only the container and are discarded when it stops. To save data, the container can mount persistent storage such as an Azure Disk or a file share (including Azure Files).</p> <p data-bbox="730 1003 1818 1203">A container builds on top of the kernel, but the kernel doesn't provide all of the APIs and services an app needs to run—most of these are provided by system files (libraries) that run above the kernel in user mode. Because a container is isolated from the host's user mode environment, the container needs its own copy of these user mode system files, which are packaged into something known as a base image. The base image serves as the foundational layer upon which your container is built, providing it with operating system services not provided by the kernel. But we'll talk more about container images later.</p> <p data-bbox="714 1227 1598 1255">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="730 272 1444 329"><h2>Containers vs. virtual machines</h2></div> <div data-bbox="730 362 1917 430"><p>In contrast to a container, a virtual machine (VMs) runs a complete operating system—including its own kernel—as shown in this diagram.</p></div> <div data-bbox="751 479 1906 829"><p>The diagram illustrates the architectural differences between containers and virtual machines. At the base is a 'Hardware' layer, represented by a box with a chip icon. Above the hardware, there are two main paths. The left path, representing containers, shows an 'OS' layer (gear icon) directly on top of the hardware, which contains 'Apps' and 'Services' boxes, all sharing a single 'Kernel' box at the bottom. The right path, representing virtual machines, shows a 'Virtual Machine' box (dashed outline) containing its own 'OS' layer (gear icon) and 'Kernel' box, which then runs 'Apps' and 'Services'. This indicates that each VM has its own complete operating system and kernel, while containers share the host's OS and kernel.</p></div> <div data-bbox="730 889 1917 998"><p>Containers and virtual machines each have their uses—in fact, many deployments of containers use virtual machines as the host operating system rather than running directly on the hardware, especially when running containers in the cloud.</p></div> <div data-bbox="714 1015 1600 1052"><p>https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p></div>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="720 264 1127 318">Container images</h2> <p data-bbox="720 354 1915 493">All containers are created from container images. A container image is a bundle of files organized into a stack of layers that resides on your local machine or in a remote container registry. The container image consists of the user mode operating system files needed to support your app, any runtimes or dependencies of your app, and any other miscellaneous configuration file your app needs to run properly.</p> <p data-bbox="720 529 1850 594">Microsoft offers several images (called base images) that you can use as a starting point to build your own container image:</p> <ul data-bbox="751 630 1915 850" style="list-style-type: none">• Windows - contains the full set of Windows APIs and system services (minus server roles).• Windows Server - contains the full set of Windows APIs and system services.• Windows Server Core - a smaller image that contains a subset of the Windows Server APIs—namely the full .NET framework. It also includes most but not all server roles (for example Fax Server is not included).• Nano Server - the smallest Windows Server image and includes support for the .NET Core APIs and some server roles. <p data-bbox="714 886 1598 911">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p> <p data-bbox="732 954 1911 1019">There are many different orchestrators that you can use with Windows containers; here are the options Microsoft provides:</p> <ul data-bbox="764 1055 1787 1117" style="list-style-type: none">• Azure Kubernetes Service (AKS) - use a managed Azure Kubernetes service• Azure Kubernetes Service (AKS) on Azure Stack HCI - use Azure Kubernetes Service on-premises <p data-bbox="714 1169 1598 1193">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p>



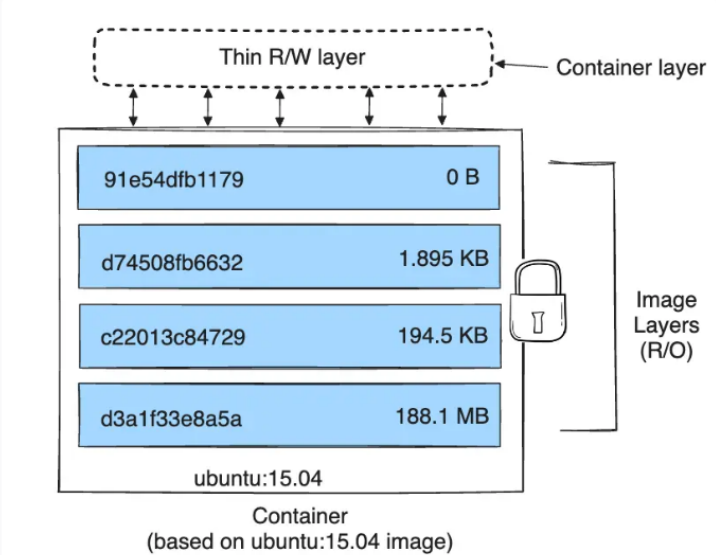
Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 1163 326">Ephemeral OS disk</h2> <p data-bbox="737 358 1913 500">By default, Azure automatically replicates the operating system disk for a virtual machine to Azure Storage to avoid data loss when the VM is relocated to another host. However, since containers aren't designed to have local state persisted, this behavior offers limited value while providing some drawbacks. These drawbacks include, but aren't limited to, slower node provisioning and higher read/write latency.</p> <p data-bbox="737 532 1822 597">By contrast, ephemeral OS disks are stored only on the host machine, just like a temporary disk. With this configuration, you get lower read/write latency, together with faster node scaling and cluster upgrades.</p> <p data-bbox="714 613 1440 643">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p> <h2 data-bbox="737 688 930 735">Volumes</h2> <p data-bbox="737 773 1860 873">Kubernetes typically treats individual pods as ephemeral, disposable resources. Applications have different approaches available to them for using and persisting data. A <i>volume</i> represents a way to store, retrieve, and persist data across pods and through the application lifecycle.</p> <p data-bbox="737 911 1898 1011">Traditional volumes are created as Kubernetes resources backed by Azure Storage. You can manually create data volumes to be assigned to pods directly or have Kubernetes automatically create them. Data volumes can use: Azure Disk, Azure Files, Azure NetApp Files, or Azure Blobs.</p> <p data-bbox="714 1027 1440 1057">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 264 1104 305">Persistent volumes</h2> <p data-bbox="730 337 1764 462">Volumes defined and created as part of the pod lifecycle only exist until you delete the pod. Pods often expect their storage to remain if a pod is rescheduled on a different host during a maintenance event, especially in StatefulSets. A <i>persistent volume</i> (PV) is a storage resource created and managed by the Kubernetes API that can exist beyond the lifetime of an individual pod.</p> <p data-bbox="730 495 1491 516">You can use the following Azure Storage services to provide the persistent volume:</p> <ul data-bbox="756 548 1008 641" style="list-style-type: none"> • Azure Disk • Azure Files • Azure Container Storage <p data-bbox="730 673 1732 727">As noted in the Volumes section, the choice of Azure Disks or Azure Files is often determined by the need for concurrent access to the data or the performance tier.</p> <div data-bbox="741 760 1766 1295"> <p>The diagram illustrates the storage architecture within an Azure Kubernetes Service (AKS) cluster. A central box labeled 'Azure Kubernetes Service (AKS) cluster' contains a blue rectangle labeled 'Persistent Volume'. Two arrows originate from this 'Persistent Volume': one points to a stack of blue and green disks labeled 'Single node/pod access' and 'Azure managed disks (Standard or Premium storage)'; the other points to a blue folder icon labeled 'Multiple concurrent node/pod access' and 'Azure Files (Standard storage)'.</p> </div> <p data-bbox="714 1328 1438 1356">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p>

Claim 1	Accused Instrumentalities
	<p>Container security protects the entire end-to-end pipeline from build to the application workloads running in Azure Kubernetes Service (AKS).</p> <p>The Secure Supply Chain includes the build environment and registry.</p> <p>Kubernetes includes security components, such as <i>pod security standards</i> and <i>Secrets</i>. Azure includes components like Active Directory, Microsoft Defender for Containers, Azure Policy, Azure Key Vault, network security groups, and orchestrated cluster upgrades. AKS combines these security components to:</p> <ul style="list-style-type: none"> • Provide a complete authentication and authorization story. • Apply AKS Built-in Azure Policy to secure your applications. • End-to-End insight from build through your application with Microsoft Defender for Containers. • Keep your AKS cluster running the latest OS security updates and Kubernetes releases. • Provide secure pod traffic and access to sensitive credentials. <p>https://learn.microsoft.com/en-us/azure/aks/concepts-security</p> <h2>Container images</h2> <p>A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p>https://kubernetes.io/docs/concepts/containers/</p> <p>6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p>Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.</p> <p>https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="722 269 1310 331">About storage drivers</h2> <p data-bbox="722 376 1871 493">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="722 558 1583 610">Storage drivers versus Docker volumes</h2> <p data-bbox="722 646 1913 896">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="722 941 1902 1058">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1373 987 1556 1013" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="722 1088 1268 1117"><a data-bbox="722 1088 1268 1117" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="735 266 1129 318">Images and layers</h2> <p data-bbox="735 354 1827 425">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="735 464 1906 805"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="735 844 1906 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="735 1153 1268 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer structure of a container. At the bottom is a stack of four blue rectangular blocks representing 'Image Layers (R/O)'. From top to bottom, they are labeled with their commit IDs and sizes: '91e54dfb1179' (0 B), 'd74508fb6632' (1.895 KB), 'c22013c84729' (194.5 KB), and 'd3a1f33e8a5a' (188.1 MB). A bracket on the right side of these blocks is labeled 'Image Layers (R/O)' and includes a padlock icon, indicating they are read-only. Above this stack is a dashed-line box labeled 'Thin R/W layer'. An arrow points from the text 'Container layer' to this dashed box. Bidirectional arrows connect the 'Thin R/W layer' to the top of the 'Image Layers (R/O)'. Below the entire stack, the text 'ubuntu:15.04' is centered, followed by 'Container (based on ubuntu:15.04 image)'.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 277 980 337">Volumes</h2> <p data-bbox="730 388 1902 509">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="714 531 1344 561">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="730 609 1266 654">Container environment</h2> <p data-bbox="730 690 1499 751">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="770 787 1476 938" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="714 971 1549 1002">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 940 331">Images</h2> <p data-bbox="737 363 1545 505">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="737 540 1551 607">You typically create a container image of your application and push it to a registry before referring to it in a <code>Pod</code>.</p> <p data-bbox="714 633 1365 662">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="730 704 978 760">Volumes</h2> <p data-bbox="730 795 1554 1209">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <code>volume</code> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="714 1235 1344 1265">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="747 277 1335 331"><h2>Open Container Initiative</h2></div> <div data-bbox="747 391 1226 435"><h3>Image Format Specification</h3></div> <div data-bbox="747 479 1890 548"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="747 581 1900 651"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="711 678 1503 743"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

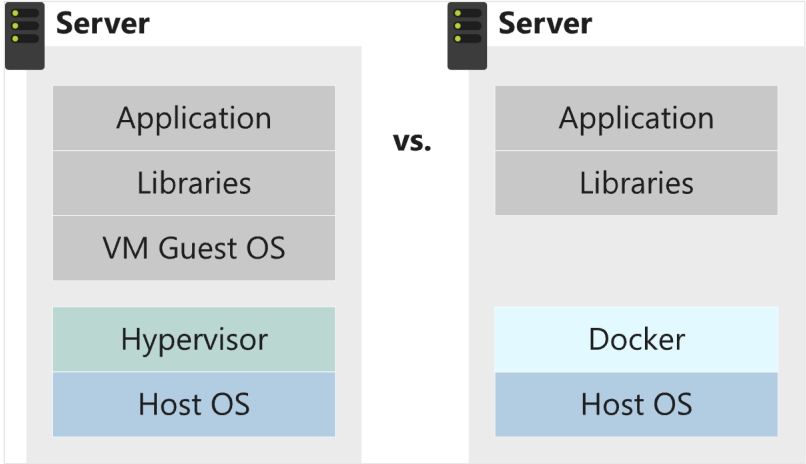
Claim 1	Accused Instrumentalities
	<p data-bbox="730 272 898 310">Overview</p> <p data-bbox="730 362 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="743 623 1906 980"> <p>The diagram illustrates the components of an image manifest. On the left, a code block for a Java class is shown: <code>public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } }</code>. An arrow points from this code to a cylinder labeled 'layer' containing the paths <code>/bin/java</code>, <code>/opt/app.jar</code>, and <code>/lib/libc</code>. This layer is then combined with an 'image index' (represented as a document icon) containing a JSON structure: <code>{ "manifests": { "platform": { "os": "linux", ... } } }</code>. Finally, this is combined with a 'config' file (also a document icon) containing a JSON structure: <code>{ ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } }</code>. Each component (layer, image index, and config) has a small square icon with the letters 'ci' in the top-left corner.</p> <p data-bbox="714 1008 1503 1073">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p> </div>

Claim 1	Accused Instrumentalities
	<h2>OCI Image Configuration</h2> <p>An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p>This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="737 272 821 306">Layer</p> <ul data-bbox="764 342 1913 667" style="list-style-type: none">• Image filesystems are composed of <i>layers</i>.• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="737 716 924 750">Image JSON</p> <ul data-bbox="764 786 1913 1110" style="list-style-type: none">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.• This JSON is considered to be immutable, because changing it would change the computed ImageID.• Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="714 1138 1528 1198">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none">• rootfs object, REQUIRED<p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p><ul style="list-style-type: none">◦ type string, REQUIRED<p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p>◦ diff_ids array of strings, REQUIRED<p>An array of layer content hashes (<code>DiffIDs</code>), in order from first to last.</p><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

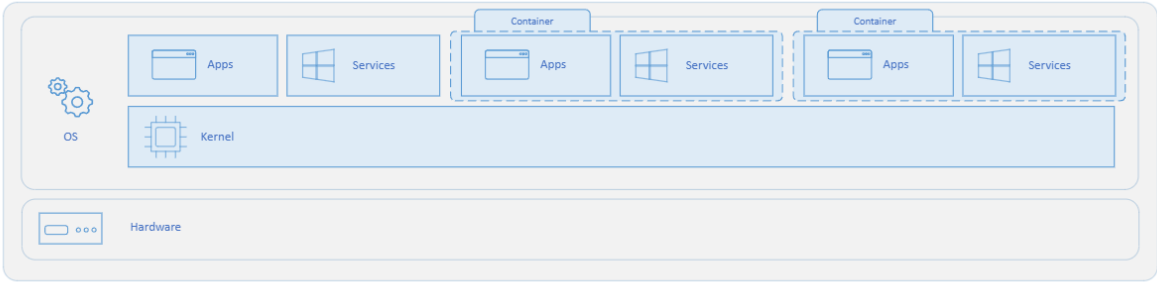
Claim 1	Accused Instrumentalities
<p>[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,</p>	<p>In the method practiced by Microsoft and/or its customer through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.</p> <p>The associated system files in the container are compatible with the host kernel, for example because they are linked against the Linux kernel and the supported host operating systems also use the Linux kernel, which has a stable binary interface. In another example, the associated system files are linked against and compatible with a Windows operating system and kernel, and the host operating system runs a compatible Windows operating system with compatible Windows kernel.</p> <p><i>See discussion in element [1a] above.</i></p> <p><i>See also, e.g.:</i></p> <p>OS</p> <p>AKS supports Ubuntu 22.04 and Azure Linux 2.0 as the node OS for Linux node pools. For Windows node pools, AKS supports Windows Server 2022 as the default OS. Windows Server 2019 is being retired after Kubernetes version 1.32 reaches end of life and isn't supported in future releases. If you need to upgrade your Windows OS version, see Upgrade from Windows Server 2019 to Windows Server 2022. For more information on using Windows Server on AKS, see Windows container considerations in Azure Kubernetes Service (AKS).</p> <p>https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts</p> <p>Container runtime</p> <p>A container runtime is software that executes containers and manages container images on a node. The runtime helps abstract away sys-calls or OS-specific functionality to run containers on Linux or Windows. For Linux node pools, containerd is used on Kubernetes version 1.19 and higher. For Windows Server 2019 and 2022 node pools, containerd is generally available and is the only runtime option on Kubernetes version 1.23 and higher.</p> <p>https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="722 264 1029 305"><h2>Docker benefits</h2></div> <div data-bbox="722 337 1533 362"><p>When we use Docker, we immediately get access to the benefits containerization offer.</p></div> <div data-bbox="722 410 1075 446"><h2>Efficient hardware use</h2></div> <div data-bbox="722 482 1915 539"><p>Containers run without using a virtual machine (VM). As we learned, the container relies on the host kernel for functions such as file system, network management, process scheduling, and memory management.</p></div> <div data-bbox="722 563 1522 1024"><p>The diagram illustrates the architectural differences between a Virtual Machine (VM) and Docker containers. On the left, labeled 'Server', the stack consists of five layers: 'Application' (grey), 'Libraries' (grey), 'VM Guest OS' (grey), 'Hypervisor' (teal), and 'Host OS' (blue). On the right, also labeled 'Server', the stack consists of three layers: 'Application' (grey), 'Libraries' (grey), and 'Docker' (light blue), which sits directly on top of the 'Host OS' (blue). A 'vs.' label is placed between the two stacks.</p></div> <div data-bbox="722 1060 1915 1153"><p>Compared to a VM, we can see that a VM requires an OS installed to provide kernel functions to the running applications inside the VM. Keep in mind that the VM OS also requires disk space, memory, and CPU time. By removing the VM and the additional OS requirement, we can free resources on the host and use it for running other containers.</p></div> <div data-bbox="714 1175 1890 1239"><p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p></div>

Claim 1	Accused Instrumentalities
	<p data-bbox="724 264 1081 302">Application portability</p> <p data-bbox="724 332 1913 391">Containers run almost everywhere: desktops, physical servers, VMs, and in the cloud. This runtime compatibility makes it easy to move containerized applications among different environments.</p> <p data-bbox="714 407 1892 472">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="724 509 1035 547">Cloud deployments</p> <p data-bbox="724 578 1919 636">Docker containers are the default container architecture the Azure containerization services use, and many other cloud platforms also support them.</p> <p data-bbox="724 667 1913 725">For instance, you can deploy Docker containers to Azure Container Instances, Azure App Service, and Azure Kubernetes Services. Each of these options provides you with different features and capabilities.</p> <p data-bbox="724 756 1913 847">For example, Azure container instances allow you to focus on designing and building your applications without the overhead of managing infrastructure. When you have many containers to orchestrate, Azure Kubernetes service makes it easy to deploy and manage large-scale container deployments.</p> <p data-bbox="714 870 1892 935">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="743 277 1549 331">Windows nodes in Kubernetes</h2> <p data-bbox="743 371 1854 488">To enable the orchestration of Windows containers in Kubernetes, include Windows nodes in your existing Linux cluster. Scheduling Windows containers in <u>Pods</u> on Kubernetes is similar to scheduling Linux-based containers.</p> <p data-bbox="743 529 1875 646">In order to run Windows containers, your Kubernetes cluster must include multiple operating systems. While you can only run the <u>control plane</u> on Linux, you can deploy worker nodes running either Windows or Linux.</p> <p data-bbox="743 686 1875 760">Windows <u>nodes</u> are supported provided that the operating system is Windows Server 2019 or Windows Server 2022.</p> <p data-bbox="743 800 1850 917">This document uses the term <i>Windows containers</i> to mean Windows containers with process isolation. Kubernetes does not support running Windows containers with Hyper-V isolation.</p> <p data-bbox="711 951 1320 980">https://kubernetes.io/docs/concepts/windows/intro/</p>

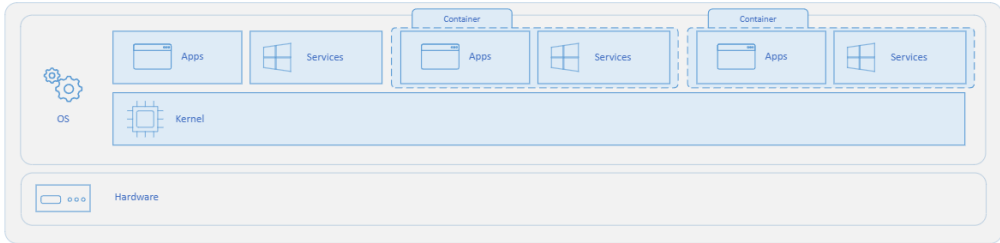
Claim 1	Accused Instrumentalities
	<h2>Windows OS version compatibility</h2> <p>On Windows nodes, strict compatibility rules apply where the host OS version must match the container base image OS version. Only Windows containers with a container operating system of Windows Server 2019 are fully supported.</p> <p>For Kubernetes v1.31, operating system compatibility for Windows nodes (and Pods) is as follows:</p> <p>Windows Server LTSC release</p> <p>Windows Server 2019</p> <p>Windows Server 2022</p> <p>Windows Server SAC release</p> <p>Windows Server version 20H2</p> <p>The Kubernetes version-skew policy also applies.</p> <p>https://kubernetes.io/docs/concepts/windows/intro/#windows-os-version-support</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 289 1218 337">How containers work</h2> <p data-bbox="730 373 1921 479">A container is an isolated, lightweight package for running an application on the host operating system. Containers build on top of the host operating system's kernel (which can be thought of as the buried plumbing of the operating system), as shown in the diagram below.</p>  <p data-bbox="730 857 1921 1039">While a container shares the host operating system's kernel, the container doesn't get unfettered access to it. Instead, the container gets an isolated—and in some cases virtualized—view of the system. For example, a container can access a virtualized version of the file system and registry, but any changes affect only the container and are discarded when it stops. To save data, the container can mount persistent storage such as an Azure Disk or a file share (including Azure Files).</p> <p data-bbox="730 1071 1921 1291">A container builds on top of the kernel, but the kernel doesn't provide all of the APIs and services an app needs to run—most of these are provided by system files (libraries) that run above the kernel in user mode. Because a container is isolated from the host's user mode environment, the container needs its own copy of these user mode system files, which are packaged into something known as a base image. The base image serves as the foundational layer upon which your container is built, providing it with operating system services not provided by the kernel. But we'll talk more about container images later.</p> <p data-bbox="730 1315 1600 1347">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p>

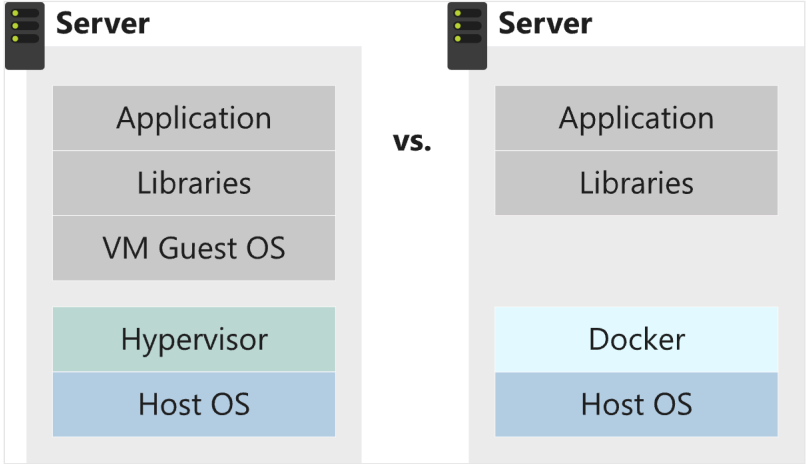
Claim 1	Accused Instrumentalities
<p>[1c] the containers of application software excluding a kernel,</p>	<p>In the method practiced by Microsoft and/or its customer through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p><i>See, e.g.:</i></p> <h2 data-bbox="722 402 1026 448">Docker benefits</h2> <p data-bbox="722 480 1530 505">When we use Docker, we immediately get access to the benefits containerization offer.</p> <h2 data-bbox="722 553 1073 589">Efficient hardware use</h2> <p data-bbox="722 623 1913 683">Containers run without using a virtual machine (VM). As we learned, the container relies on the host kernel for functions such as file system, network management, process scheduling, and memory management.</p> <div data-bbox="722 704 1520 1167"> <p>The diagram illustrates the architectural differences between a Virtual Machine (VM) and Docker containers. On the left, labeled 'Server', the VM architecture is shown as a stack: 'Application' and 'Libraries' sit on top of a 'VM Guest OS', which runs on a 'Hypervisor' layer, which in turn runs on the 'Host OS'. On the right, also labeled 'Server', the Docker architecture is shown as a stack: 'Application' and 'Libraries' sit on top of a 'Docker' layer, which runs directly on the 'Host OS'. The two architectures are compared with 'vs.' between them.</p> </div> <p data-bbox="722 1203 1913 1295">Compared to a VM, we can see that a VM requires an OS installed to provide kernel functions to the running applications inside the VM. Keep in mind that the VM OS also requires disk space, memory, and CPU time. By removing the VM and the additional OS requirement, we can free resources on the host and use it for running other containers.</p> <p data-bbox="722 1317 1892 1382">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 261 1031 293">Container isolation</p> <p data-bbox="730 329 1915 422">Docker containers provide security features to run multiple containers simultaneously on the same host without affecting each other. As we learned, we can configure both data storage and network configuration to isolate our containers or share data and connectivity between specific containers.</p> <p data-bbox="730 453 1102 477">Let's compare this feature to using VMs.</p> <div data-bbox="730 500 1524 873"> <p>The diagram illustrates two architectural models for running applications on a server. The left model, labeled 'Server', shows three separate Virtual Machines (VMs) stacked vertically. Each VM contains its own 'VM OS', 'Libs' (libraries), and 'App' (application). These VMs are managed by a 'Hypervisor' layer, which sits on top of the 'Host OS'. The right model, also labeled 'Server', shows three containers stacked vertically. Each container contains 'Libs' and an 'App'. These containers are managed by a 'Docker' layer, which sits on top of the 'Host OS'. The two models are compared with 'vs.' between them.</p> </div> <p data-bbox="730 907 1915 1000">Assume we have a physical host running two VMs. We have three applications that we want to run isolated from each other. We decide to deploy the first app onto VM1 and the second onto VM2 to separate the two apps from each other. If we now choose to install the third application, we'll need to install another VM to continue this pattern.</p> <p data-bbox="714 1016 1890 1081">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="730 1112 1081 1149">Application portability</p> <p data-bbox="730 1182 1915 1240">Containers run almost everywhere: desktops, physical servers, VMs, and in the cloud. This runtime compatibility makes it easy to move containerized applications among different environments.</p> <p data-bbox="714 1256 1890 1321">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

Claim 1	Accused Instrumentalities
	<p>Cloud deployments</p> <p>Docker containers are the default container architecture the Azure containerization services use, and many other cloud platforms also support them.</p> <p>For instance, you can deploy Docker containers to Azure Container Instances, Azure App Service, and Azure Kubernetes Services. Each of these options provides you with different features and capabilities.</p> <p>For example, Azure container instances allow you to focus on designing and building your applications without the overhead of managing infrastructure. When you have many containers to orchestrate, Azure Kubernetes service makes it easy to deploy and manage large-scale container deployments.</p> <p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

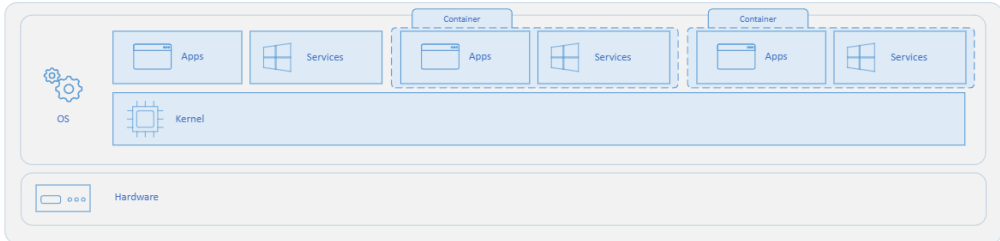
Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 280 1150 326">How containers work</h2> <p data-bbox="730 354 1759 444">A container is an isolated, lightweight package for running an application on the host operating system. Containers build on top of the host operating system's kernel (which can be thought of as the buried plumbing of the operating system), as shown in the diagram below.</p>  <p data-bbox="730 776 1759 932">While a container shares the host operating system's kernel, the container doesn't get unfettered access to it. Instead, the container gets an isolated—and in some cases virtualized—view of the system. For example, a container can access a virtualized version of the file system and registry, but any changes affect only the container and are discarded when it stops. To save data, the container can mount persistent storage such as an Azure Disk or a file share (including Azure Files).</p> <p data-bbox="730 959 1759 1149">A container builds on top of the kernel, but the kernel doesn't provide all of the APIs and services an app needs to run—most of these are provided by system files (libraries) that run above the kernel in user mode. Because a container is isolated from the host's user mode environment, the container needs its own copy of these user mode system files, which are packaged into something known as a base image. The base image serves as the foundational layer upon which your container is built, providing it with operating system services not provided by the kernel. But we'll talk more about container images later.</p> <p data-bbox="730 1177 1600 1203">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p> <p data-bbox="730 1230 1444 1256">6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p data-bbox="730 1284 1688 1369">Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.</p> <p data-bbox="730 1380 1810 1406">https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p>

Claim 1	Accused Instrumentalities
<p>[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by Microsoft and/or its customer through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.</p> <p>For example, each container will utilize its own associated system files, including runtime linked libraries (<i>e.g.</i> .dll/.so files), configuration files, etc., not the corresponding associated local system files (<i>e.g.</i>, libraries and configuration files of the host OS). As described above and below, in the Accused Instrumentalities the associated system files provide at least some of the same functionalities as the associated local system files. The host/node's associated local system files remain resident on the host/node, for example for use by system processes or applications outside the container environment.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<div data-bbox="722 264 1026 305"><h2>Docker benefits</h2></div> <div data-bbox="722 337 1530 362"><p>When we use Docker, we immediately get access to the benefits containerization offer.</p></div> <div data-bbox="722 410 1073 446"><h2>Efficient hardware use</h2></div> <div data-bbox="722 482 1913 539"><p>Containers run without using a virtual machine (VM). As we learned, the container relies on the host kernel for functions such as file system, network management, process scheduling, and memory management.</p></div> <div data-bbox="722 563 1522 1024"><p>The diagram illustrates the architectural differences between a Virtual Machine (VM) and Docker containers. On the left, labeled 'Server', the stack consists of five layers: 'Application' (grey), 'Libraries' (grey), 'VM Guest OS' (grey), 'Hypervisor' (teal), and 'Host OS' (blue). On the right, also labeled 'Server', the stack consists of three layers: 'Application' (grey), 'Libraries' (grey), and 'Docker' (light blue), which sits directly on top of the 'Host OS' (blue). A 'vs.' label is placed between the two stacks.</p></div> <div data-bbox="722 1060 1913 1153"><p>Compared to a VM, we can see that a VM requires an OS installed to provide kernel functions to the running applications inside the VM. Keep in mind that the VM OS also requires disk space, memory, and CPU time. By removing the VM and the additional OS requirement, we can free resources on the host and use it for running other containers.</p></div> <div data-bbox="714 1175 1890 1239"><p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p></div>

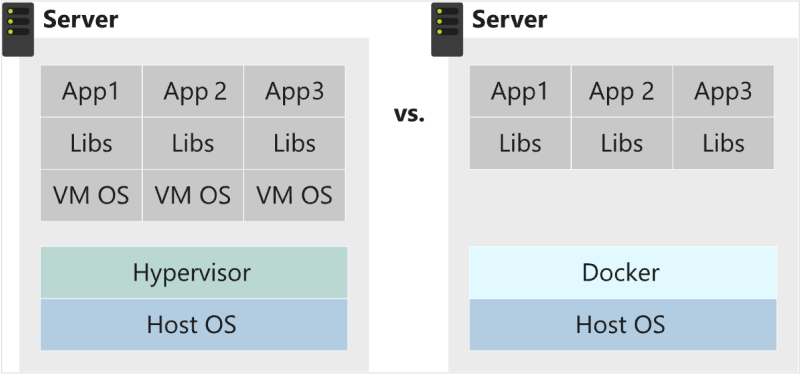
Claim 1	Accused Instrumentalities
	<p data-bbox="730 261 1031 293">Container isolation</p> <p data-bbox="730 329 1915 422">Docker containers provide security features to run multiple containers simultaneously on the same host without affecting each other. As we learned, we can configure both data storage and network configuration to isolate our containers or share data and connectivity between specific containers.</p> <p data-bbox="730 453 1102 477">Let's compare this feature to using VMs.</p> <div data-bbox="730 500 1524 873"> </div> <p data-bbox="730 907 1915 1000">Assume we have a physical host running two VMs. We have three applications that we want to run isolated from each other. We decide to deploy the first app onto VM1 and the second onto VM2 to separate the two apps from each other. If we now choose to install the third application, we'll need to install another VM to continue this pattern.</p> <p data-bbox="714 1016 1890 1081">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="730 1112 1081 1144">Application portability</p> <p data-bbox="730 1180 1915 1240">Containers run almost everywhere: desktops, physical servers, VMs, and in the cloud. This runtime compatibility makes it easy to move containerized applications among different environments.</p> <p data-bbox="714 1256 1890 1321">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

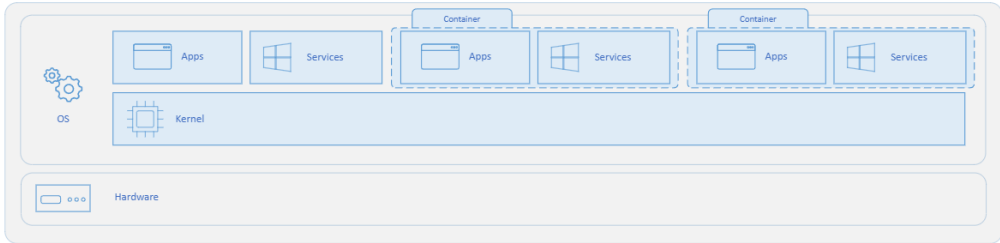
Claim 1	Accused Instrumentalities
	<p data-bbox="726 269 1037 310">Cloud deployments</p> <p data-bbox="726 339 1919 399">Docker containers are the default container architecture the Azure containerization services use, and many other cloud platforms also support them.</p> <p data-bbox="726 428 1919 488">For instance, you can deploy Docker containers to Azure Container Instances, Azure App Service, and Azure Kubernetes Services. Each of these options provides you with different features and capabilities.</p> <p data-bbox="726 518 1919 610">For example, Azure container instances allow you to focus on designing and building your applications without the overhead of managing infrastructure. When you have many containers to orchestrate, Azure Kubernetes service makes it easy to deploy and manage large-scale container deployments.</p> <p data-bbox="714 631 1890 691">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="726 721 1068 761">Container runtime</p> <p data-bbox="726 807 1919 948">A container runtime is software that executes containers and manages container images on a node. The runtime helps abstract away sys-calls or OS-specific functionality to run containers on Linux or Windows. For Linux node pools, containerd is used on Kubernetes version 1.19 and higher. For Windows Server 2019 and 2022 node pools, containerd is generally available and is the only runtime option on Kubernetes version 1.23 and higher.</p> <p data-bbox="714 969 1455 1003">https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 277 1150 321">How containers work</h2> <p data-bbox="730 354 1757 443">A container is an isolated, lightweight package for running an application on the host operating system. Containers build on top of the host operating system's kernel (which can be thought of as the buried plumbing of the operating system), as shown in the diagram below.</p>  <p data-bbox="730 776 1757 930">While a container shares the host operating system's kernel, the container doesn't get unfettered access to it. Instead, the container gets an isolated—and in some cases virtualized—view of the system. For example, a container can access a virtualized version of the file system and registry, but any changes affect only the container and are discarded when it stops. To save data, the container can mount persistent storage such as an Azure Disk or a file share (including Azure Files).</p> <p data-bbox="730 963 1757 1149">A container builds on top of the kernel, but the kernel doesn't provide all of the APIs and services an app needs to run—most of these are provided by system files (libraries) that run above the kernel in user mode. Because a container is isolated from the host's user mode environment, the container needs its own copy of these user mode system files, which are packaged into something known as a base image. The base image serves as the foundational layer upon which your container is built, providing it with operating system services not provided by the kernel. But we'll talk more about container images later.</p> <p data-bbox="730 1174 1598 1203">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p> <p data-bbox="730 1230 1442 1255">6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p data-bbox="730 1287 1686 1369">Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.</p> <p data-bbox="730 1385 1808 1414">https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p>

Claim 1	Accused Instrumentalities
<p>[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by Microsoft and/or its customer through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p>For example, in some cases the host OS and container will use one or more identical system files, for example when both the host and the container incorporate the same Linux distribution version, or when both host and container use the same Windows services. In the case where the associated system files are identical to the associated local system files, they are copies thereof. In other cases modified copies are used instead, for example when different versions of the same library, or configuration files with different parameters, are used by the host and container.</p> <p><i>See discussion in elements [1a], [1b] above.</i></p>

Claim 1	Accused Instrumentalities
<p>[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,</p>	<p>In the method practiced by Microsoft and/or its customer through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p>For example, each container has an isolated runtime environment that cannot be accessed by other containers, for example including a per-container writeable layer or other ephemeral per-container storage. For another example, when the plurality of secure containers each corresponds to a different container image, each container cannot access another container's image and therefore application software.</p> <p><i>See, e.g.:</i></p>

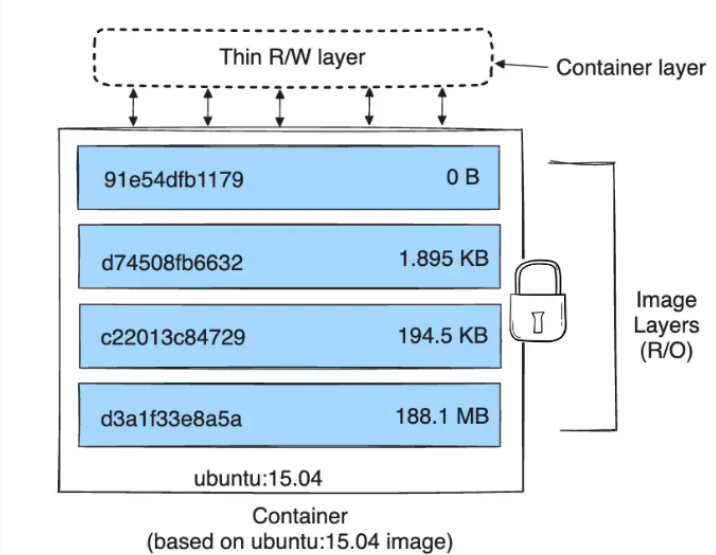
Claim 1	Accused Instrumentalities
	<div><h3>Container isolation</h3><p>Docker containers provide security features to run multiple containers simultaneously on the same host without affecting each other. As we learned, we can configure both data storage and network configuration to isolate our containers or share data and connectivity between specific containers.</p><p>Let's compare this feature to using VMs.</p><p>Assume we have a physical host running two VMs. We have three applications that we want to run isolated from each other. We decide to deploy the first app onto VM1 and the second onto VM2 to separate the two apps from each other. If we now choose to install the third application, we'll need to install another VM to continue this pattern.</p><p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p></div>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 280 1150 326">How containers work</h2> <p data-bbox="730 354 1755 444">A container is an isolated, lightweight package for running an application on the host operating system. Containers build on top of the host operating system's kernel (which can be thought of as the buried plumbing of the operating system), as shown in the diagram below.</p>  <p data-bbox="730 776 1755 932">While a container shares the host operating system's kernel, the container doesn't get unfettered access to it. Instead, the container gets an isolated—and in some cases virtualized—view of the system. For example, a container can access a virtualized version of the file system and registry, but any changes affect only the container and are discarded when it stops. To save data, the container can mount persistent storage such as an Azure Disk or a file share (including Azure Files).</p> <p data-bbox="730 959 1755 1149">A container builds on top of the kernel, but the kernel doesn't provide all of the APIs and services an app needs to run—most of these are provided by system files (libraries) that run above the kernel in user mode. Because a container is isolated from the host's user mode environment, the container needs its own copy of these user mode system files, which are packaged into something known as a base image. The base image serves as the foundational layer upon which your container is built, providing it with operating system services not provided by the kernel. But we'll talk more about container images later.</p> <p data-bbox="714 1174 1598 1203">https://learn.microsoft.com/en-us/virtualization/windowscontainers/about/</p>

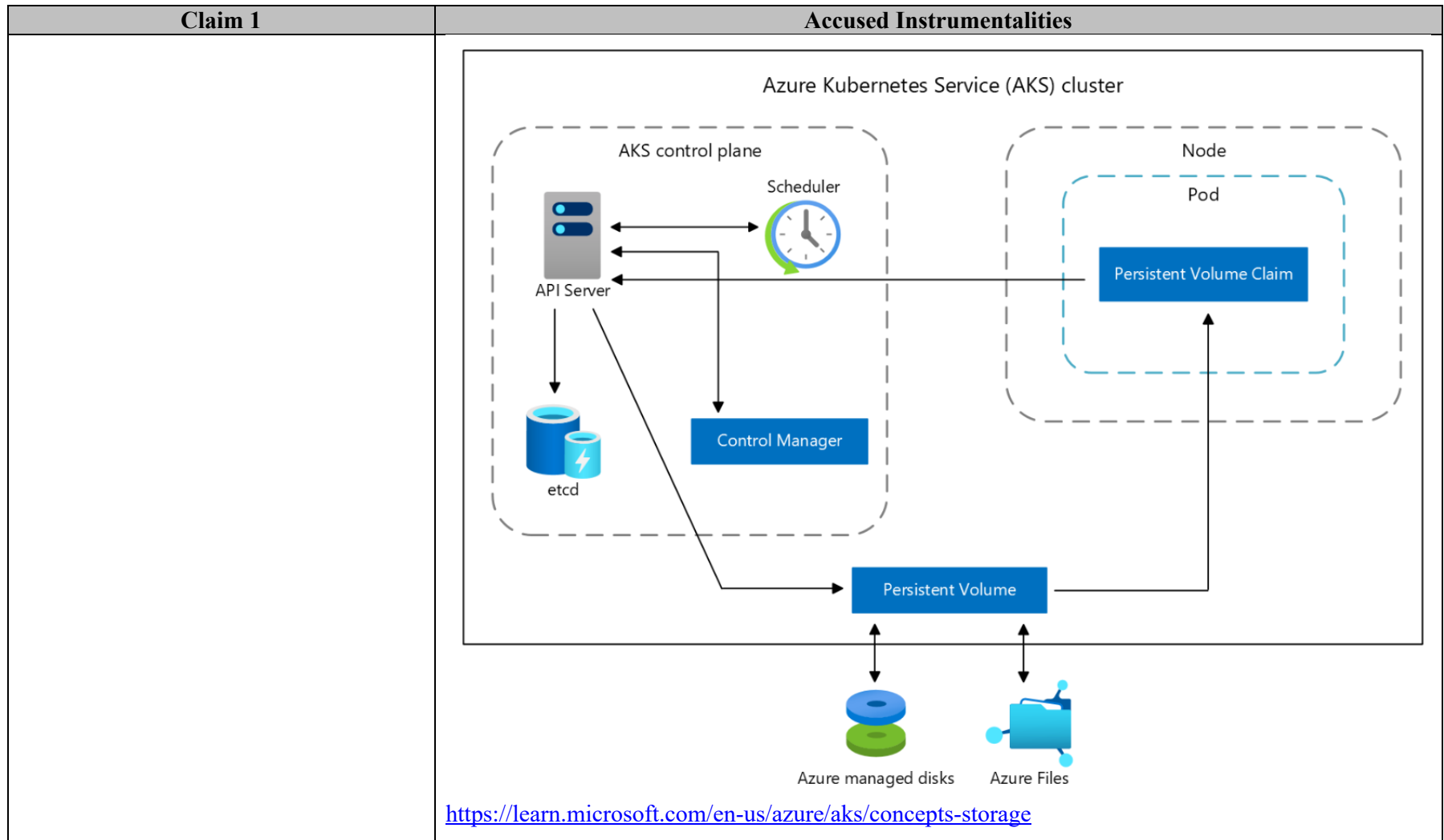
Claim 1	Accused Instrumentalities
	<p>Container security protects the entire end-to-end pipeline from build to the application workloads running in Azure Kubernetes Service (AKS).</p> <p>The Secure Supply Chain includes the build environment and registry.</p> <p>Kubernetes includes security components, such as <i>pod security standards</i> and <i>Secrets</i>. Azure includes components like Active Directory, Microsoft Defender for Containers, Azure Policy, Azure Key Vault, network security groups, and orchestrated cluster upgrades. AKS combines these security components to:</p> <ul style="list-style-type: none">• Provide a complete authentication and authorization story.• Apply AKS Built-in Azure Policy to secure your applications.• End-to-End insight from build through your application with Microsoft Defender for Containers.• Keep your AKS cluster running the latest OS security updates and Kubernetes releases.• Provide secure pod traffic and access to sensitive credentials. <p>https://learn.microsoft.com/en-us/azure/aks/concepts-security</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="722 269 1312 334">About storage drivers</h2> <p data-bbox="722 376 1871 493">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="722 558 1583 610">Storage drivers versus Docker volumes</h2> <p data-bbox="722 646 1913 896">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="722 941 1902 1058">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1373 987 1556 1011" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="722 1088 1266 1118"><a data-bbox="722 1088 1266 1118" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="732 266 1131 318">Images and layers</h2> <p data-bbox="732 354 1827 423">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="751 488 1480 781"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="732 846 1900 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="714 1154 1268 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer structure of a container. At the bottom is a stack of four blue rectangular blocks representing image layers, labeled with their IDs and sizes: <code>91e54dfb1179</code> (0 B), <code>d74508fb6632</code> (1.895 KB), <code>c22013c84729</code> (194.5 KB), and <code>d3a1f33e8a5a</code> (188.1 MB). A bracket on the right side of these blocks is labeled "Image Layers (R/O)" and has a padlock icon next to it, indicating they are read-only. Above this stack is a dashed rectangular box labeled "Thin R/W layer". A bracket on the right side of this dashed box is labeled "Container layer". Arrows point from the top of each image layer to the bottom of the thin R/W layer, showing that the container layer is built upon the image layers. Below the entire stack, the text "Container (based on ubuntu:15.04 image)" is displayed.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by Microsoft and/or its customer through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p>For example, the container's root file system comprises the image layer(s), an ephemeral writeable layer (e.g., in Docker terminology the container layer), and optionally one or more volumes. This root file system is distinct and isolated from the host operating system's root file system.</p> <p><i>See, e.g.:</i></p>

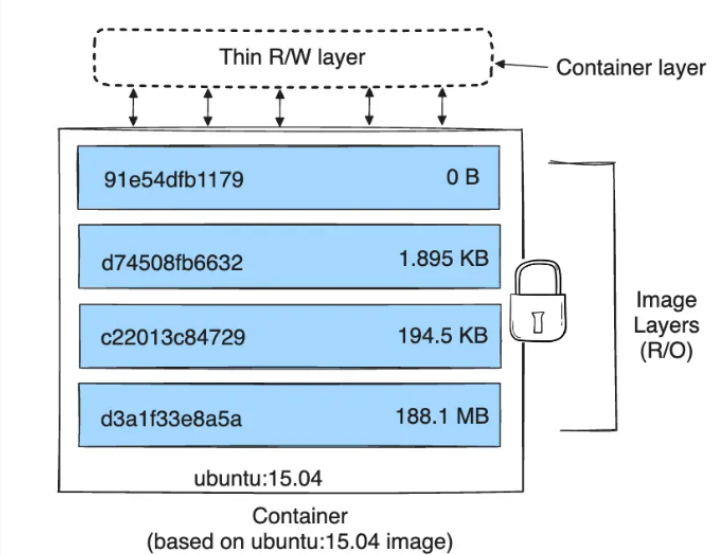


Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 1163 326">Ephemeral OS disk</h2> <p data-bbox="737 358 1913 500">By default, Azure automatically replicates the operating system disk for a virtual machine to Azure Storage to avoid data loss when the VM is relocated to another host. However, since containers aren't designed to have local state persisted, this behavior offers limited value while providing some drawbacks. These drawbacks include, but aren't limited to, slower node provisioning and higher read/write latency.</p> <p data-bbox="737 532 1822 597">By contrast, ephemeral OS disks are stored only on the host machine, just like a temporary disk. With this configuration, you get lower read/write latency, together with faster node scaling and cluster upgrades.</p> <p data-bbox="714 613 1438 643">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p> <h2 data-bbox="737 688 930 742">Volumes</h2> <p data-bbox="737 774 1860 873">Kubernetes typically treats individual pods as ephemeral, disposable resources. Applications have different approaches available to them for using and persisting data. A <i>volume</i> represents a way to store, retrieve, and persist data across pods and through the application lifecycle.</p> <p data-bbox="737 906 1898 1011">Traditional volumes are created as Kubernetes resources backed by Azure Storage. You can manually create data volumes to be assigned to pods directly or have Kubernetes automatically create them. Data volumes can use: Azure Disk, Azure Files, Azure NetApp Files, or Azure Blobs.</p> <p data-bbox="714 1027 1438 1057">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 264 1104 305">Persistent volumes</h2> <p data-bbox="730 337 1764 462">Volumes defined and created as part of the pod lifecycle only exist until you delete the pod. Pods often expect their storage to remain if a pod is rescheduled on a different host during a maintenance event, especially in StatefulSets. A <i>persistent volume</i> (PV) is a storage resource created and managed by the Kubernetes API that can exist beyond the lifetime of an individual pod.</p> <p data-bbox="730 495 1491 516">You can use the following Azure Storage services to provide the persistent volume:</p> <ul data-bbox="756 548 1003 638" style="list-style-type: none">• Azure Disk• Azure Files• Azure Container Storage <p data-bbox="730 670 1732 727">As noted in the Volumes section, the choice of Azure Disks or Azure Files is often determined by the need for concurrent access to the data or the performance tier.</p> <div data-bbox="741 760 1766 1295"><p>The diagram illustrates the connection between an Azure Kubernetes Service (AKS) cluster and Azure storage services. A rounded rectangle labeled 'Azure Kubernetes Service (AKS) cluster' contains a blue box labeled 'Persistent Volume'. Two arrows originate from this 'Persistent Volume' box. One arrow points to a stack of two disks (one blue, one green) labeled 'Azure managed disks (Standard or Premium storage)', with the text 'Single node/pod access' above it. The other arrow points to a blue folder icon labeled 'Azure Files (Standard storage)', with the text 'Multiple concurrent node/pod access' above it.</p></div> <p data-bbox="714 1328 1440 1356">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 1814 326">What kind of disks are supported for Windows?</h2> <p data-bbox="737 358 1871 423">Azure Disks and Azure Files are the supported volume types, and are accessed as New Technology File System (NTFS) volumes in the Windows Server container.</p> <p data-bbox="711 448 1394 480">https://learn.microsoft.com/en-us/azure/aks/windows-faq</p> <h2 data-bbox="720 516 1312 581">About storage drivers</h2> <p data-bbox="720 623 1871 740">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="720 805 1585 859">Storage drivers versus Docker volumes</h2> <p data-bbox="720 893 1913 1143">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="720 1190 1902 1307">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="711 1333 1268 1365">https://docs.docker.com/storage/storagedriver/</p>




Claim 1	Accused Instrumentalities
	<h2 data-bbox="735 266 1129 318">Images and layers</h2> <p data-bbox="735 354 1827 425">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="735 464 1906 805"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="735 844 1906 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="735 1153 1268 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer architecture of a Docker container. At the bottom, a stack of four blue rectangular blocks represents the 'Image Layers (R/O)'. Each block contains a hexadecimal hash and a size: '91e54dfb1179' (0 B), 'd74508fb6632' (1.895 KB), 'c22013c84729' (194.5 KB), and 'd3a1f33e8a5a' (188.1 MB). A bracket on the right side of this stack is labeled 'Image Layers (R/O)' and includes a padlock icon, indicating they are read-only. Above this stack is a dashed-line box labeled 'Thin R/W layer'. A bracket on the right side of this dashed box is labeled 'Container layer'. Arrows point from the top of each image layer to the thin R/W layer. Below the image layers, the text 'ubuntu:15.04' is displayed, followed by 'Container (based on ubuntu:15.04 image)'.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 277 980 337">Volumes</h2> <p data-bbox="730 388 1902 509">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="714 531 1344 561">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="730 609 1266 654">Container environment</h2> <p data-bbox="730 690 1499 751">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="770 787 1476 938" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="714 971 1549 1002">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 940 331">Images</h2> <p data-bbox="737 362 1545 505">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="737 537 1551 607">You typically create a container image of your application and push it to a registry before referring to it in a Pod.</p> <p data-bbox="714 633 1365 662">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="730 703 980 761">Volumes</h2> <p data-bbox="730 792 1554 1209">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a Pod and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested.</p> <p data-bbox="714 1235 1344 1265">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="747 272 1335 331"><h2>Open Container Initiative</h2></div> <div data-bbox="747 386 1226 435"><h3>Image Format Specification</h3></div> <div data-bbox="747 475 1890 548"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="747 578 1900 651"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="709 675 1503 740"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 272 898 310">Overview</p> <p data-bbox="730 362 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="743 623 1906 980"> <pre data-bbox="743 753 1037 834"> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p data-bbox="1058 786 1100 802">→</p> <div data-bbox="1121 623 1906 980"> <div data-bbox="1121 623 1346 948">  <p data-bbox="1184 753 1325 834">/bin/java /opt/app.jar /lib/libc</p> <p data-bbox="1226 948 1283 980">layer</p> </div> <div data-bbox="1367 786 1398 810">+</div> <div data-bbox="1398 623 1633 948">  <pre data-bbox="1419 704 1612 850"> { "manifests": { "platform": { "os": "linux", ... } } } </pre> <p data-bbox="1457 948 1598 980">image index</p> </div> <div data-bbox="1646 786 1677 810">+</div> <div data-bbox="1677 623 1906 948">  <pre data-bbox="1698 672 1877 883"> { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } } </pre> <p data-bbox="1772 948 1829 980">config</p> </div> </div> </div> <p data-bbox="714 1008 1503 1073"> https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md </p>

Claim 1	Accused Instrumentalities
	<h2>OCI Image Configuration</h2> <p>An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p>This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="741 274 821 306">Layer</p> <ul data-bbox="766 342 1917 667" style="list-style-type: none">• Image filesystems are composed of <i>layers</i>.• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="741 716 924 748">Image JSON</p> <ul data-bbox="766 784 1917 1109" style="list-style-type: none">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.• This JSON is considered to be immutable, because changing it would change the computed ImageID.• Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="716 1138 1528 1198">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none">• rootfs <i>object</i>, REQUIRED<p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p><ul style="list-style-type: none">◦ type <i>string</i>, REQUIRED<p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p>◦ diff_ids <i>array of strings</i>, REQUIRED<p>An array of layer content hashes (<code>DiffIDs</code>), in order from first to last.</p><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Exhibit 3



US007784058B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,784,058 B2**
(45) **Date of Patent:** **Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES**

2002/0004854 A1 1/2002 Hartley
2002/0174215 A1 11/2002 Schaefer 709/224
2003/0101292 A1 5/2003 Fisher
2004/0025165 A1* 2/2004 Desoli et al. 719/310

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul
O’Leary**, Kanata (CA); **Dean Huffman**,
Kanata (CA)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002
WO WO 2006/039181 A 4/2006

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

OTHER PUBLICATIONS

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 1293 days.

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

* cited by examiner

(21) Appl. No.: **10/946,536**

Primary Examiner—Hyung S Sough

Assistant Examiner—Syed Roni

(22) Filed: **Sep. 21, 2004**

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt,
Milbrath & Gilchrist, P.A. Attorneys at Law

(65) **Prior Publication Data**
US 2005/0066303 A1 Mar. 24, 2005

(57) **ABSTRACT**

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep.
22, 2003.

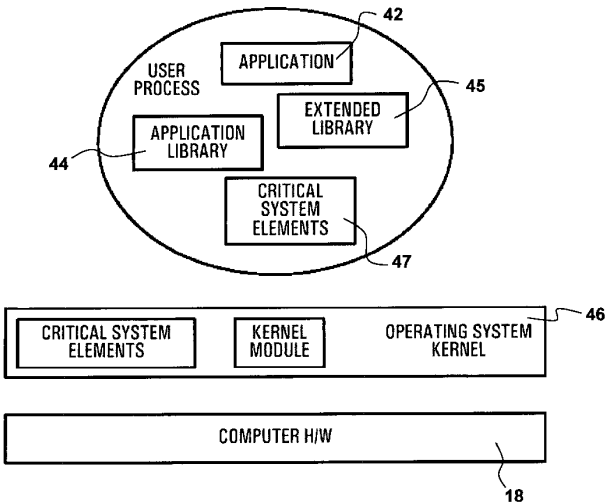
A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

(51) **Int. Cl.**
G06F 9/22 (2006.01)
(52) **U.S. Cl.** **719/310**; 719/319
(58) **Field of Classification Search** 719/310,
719/319
See application file for complete search history.

(56) **References Cited**
U.S. PATENT DOCUMENTS

5,481,706 A * 1/1996 Peek 710/200
6,212,574 B1 * 4/2001 O’Rourke et al. 719/321
6,260,075 B1 * 7/2001 Cabrero et al. 719/310

18 Claims, 7 Drawing Sheets



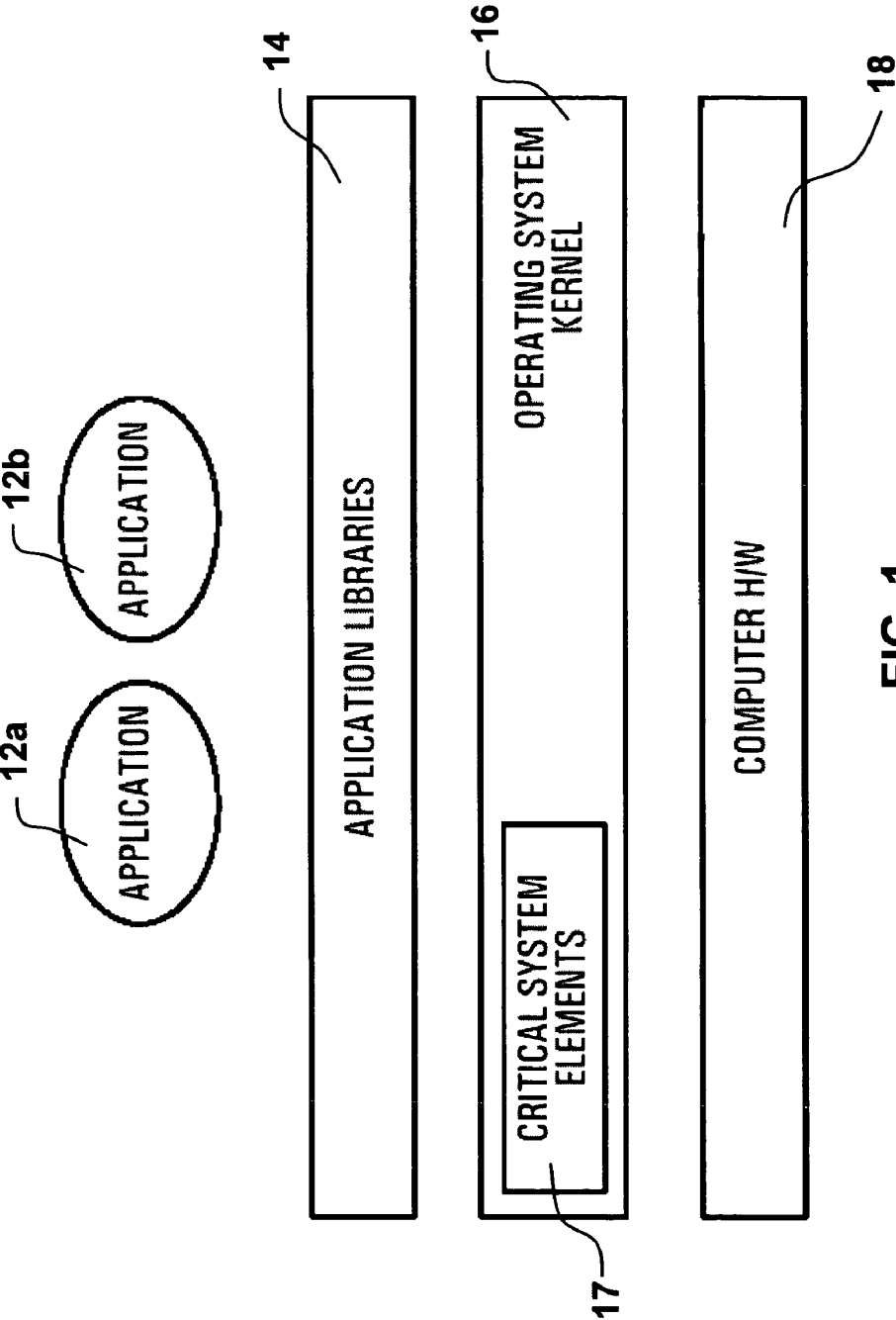


FIG. 1
Prior Art

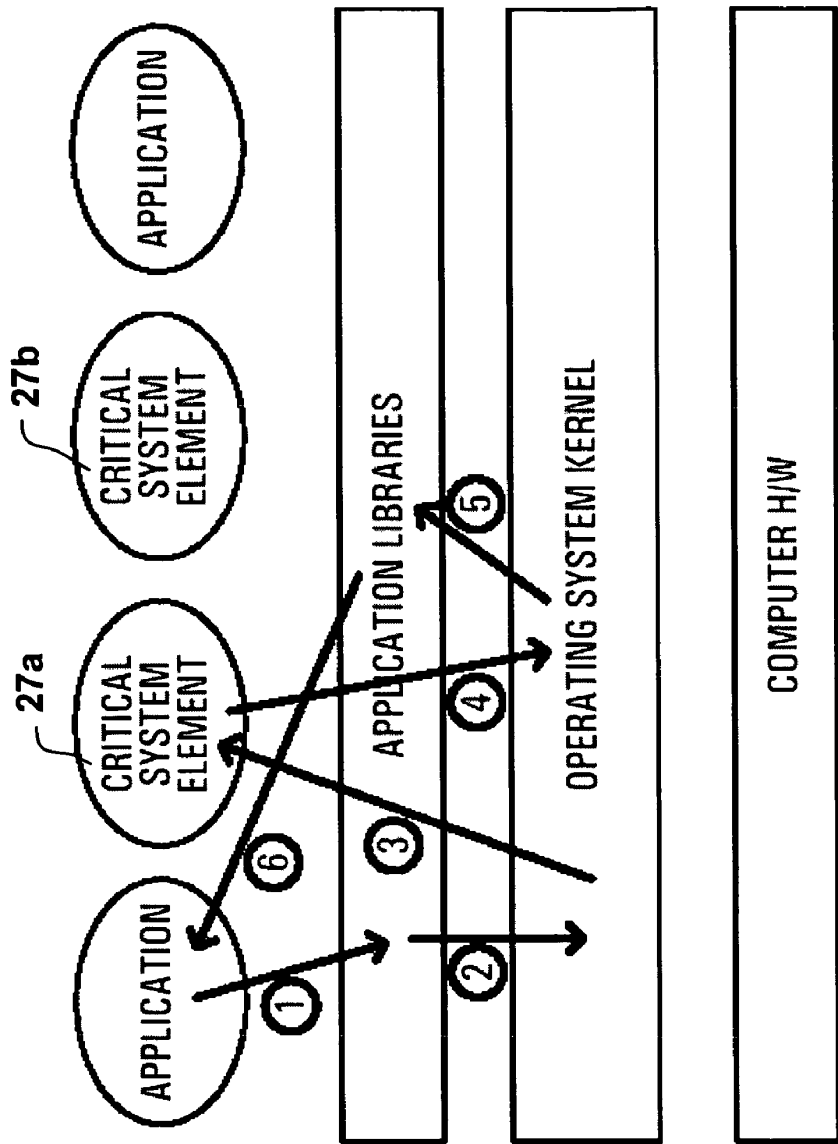


FIG. 2a
Prior Art

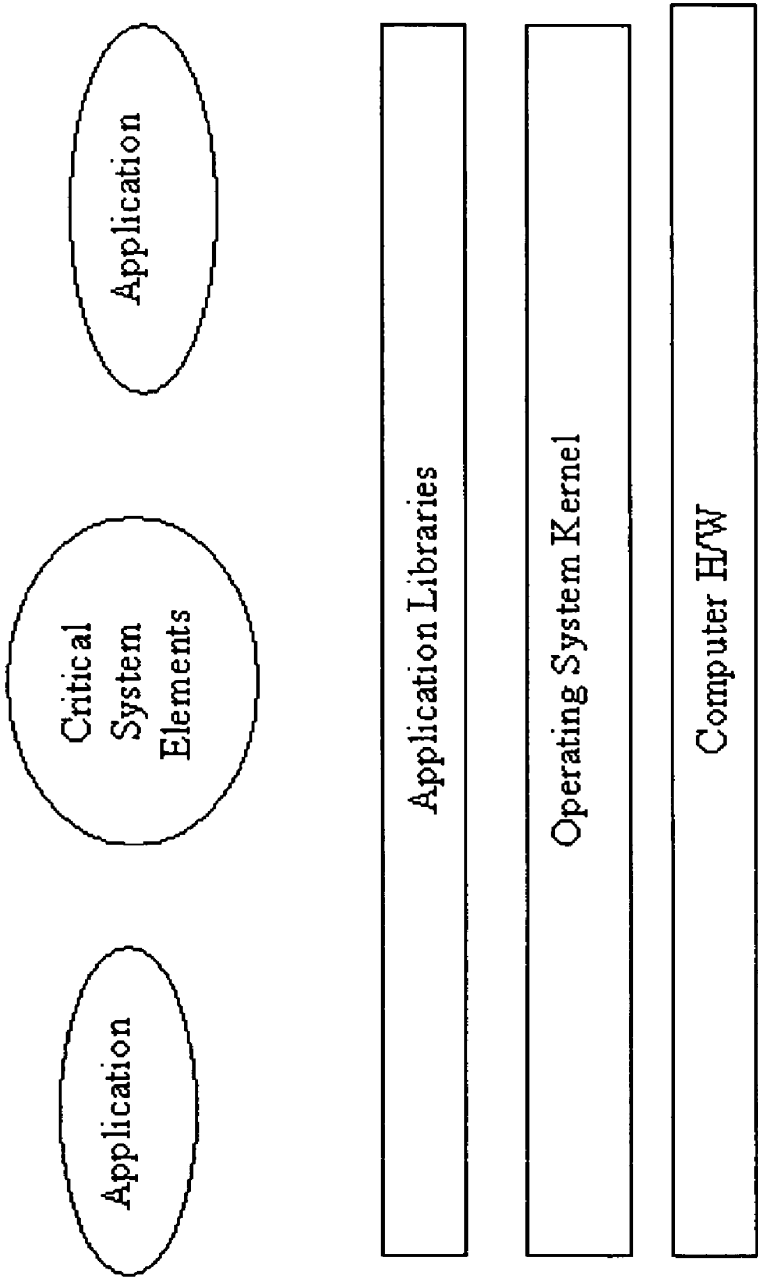


FIG. 2b
Prior Art

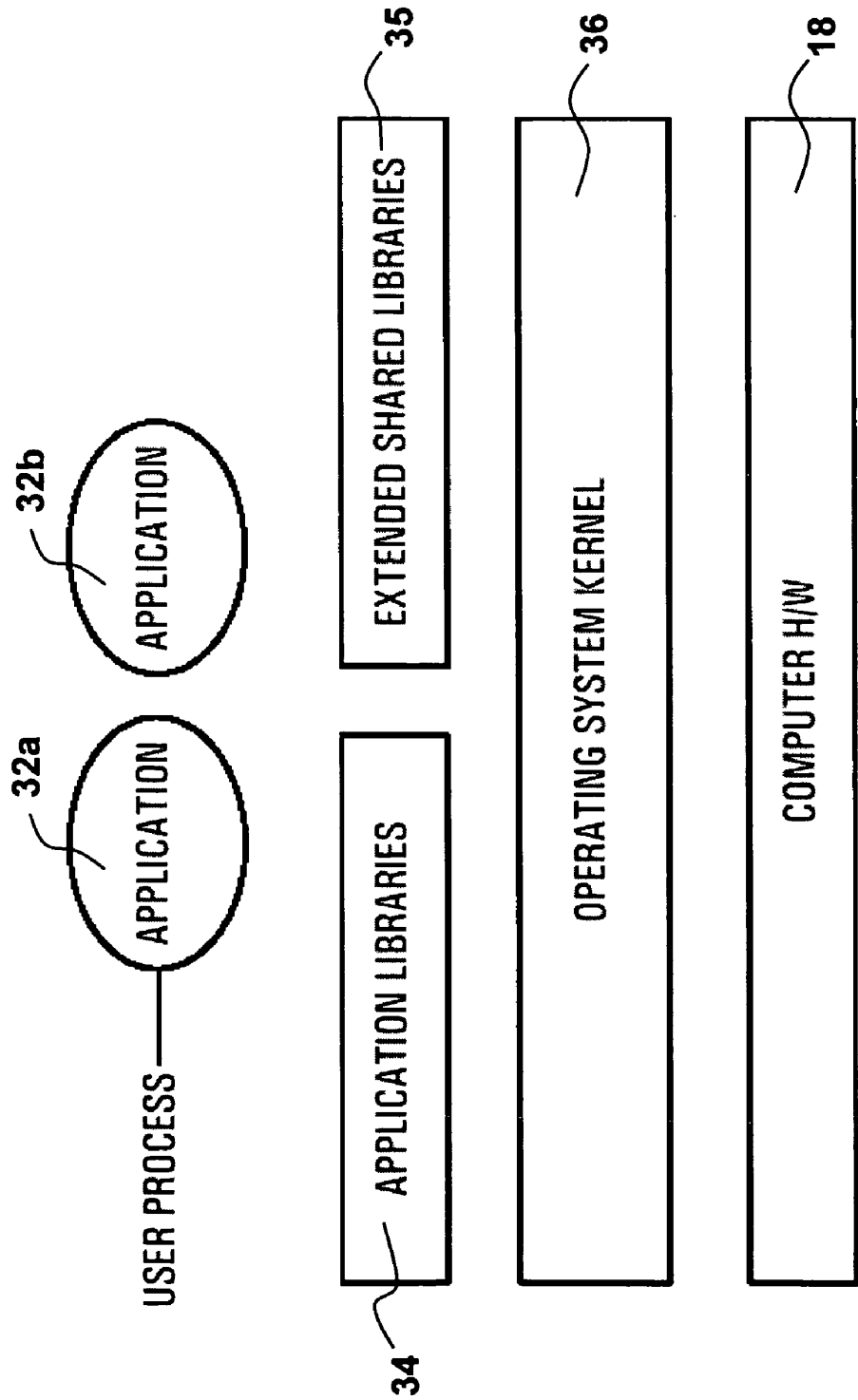
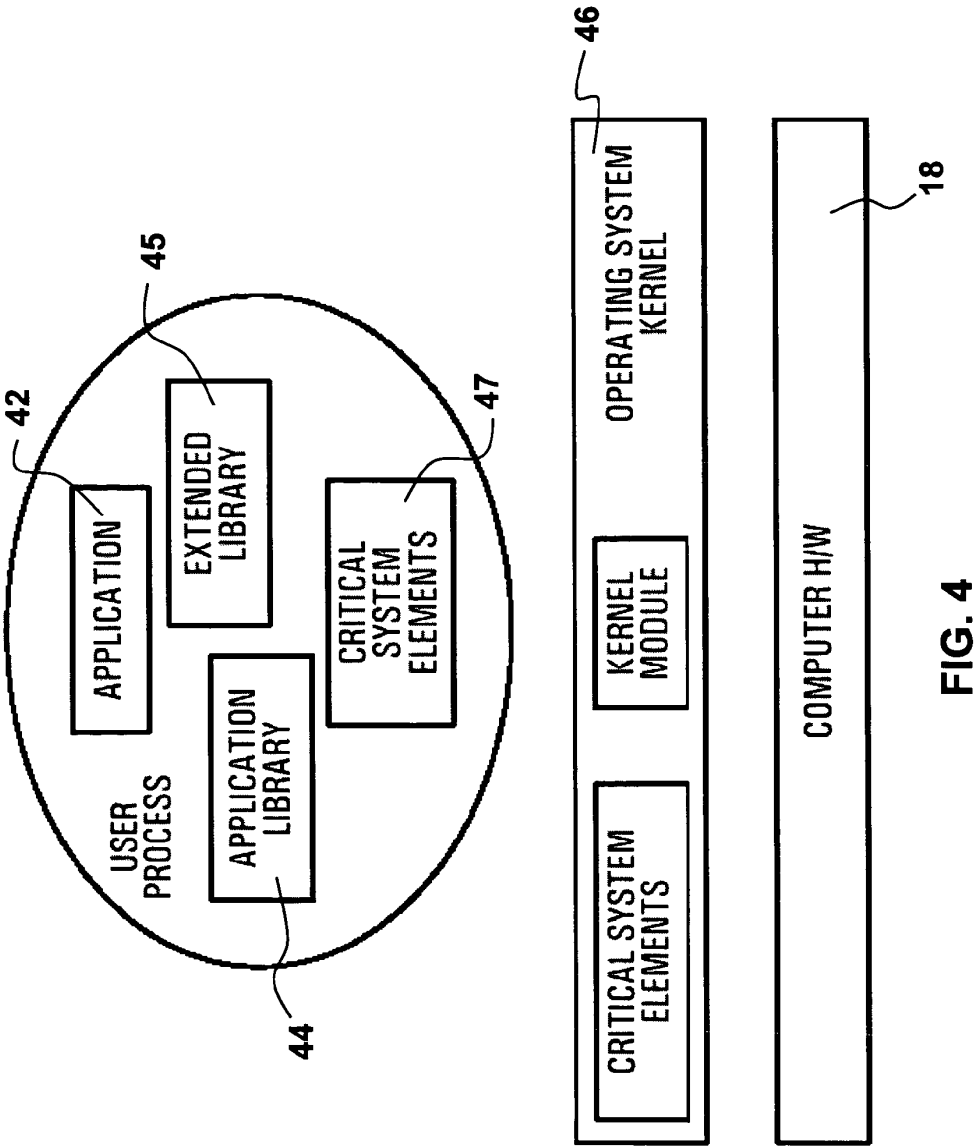


FIG. 3



U.S. Patent

Aug. 24, 2010

Sheet 6 of 7

US 7,784,058 B2

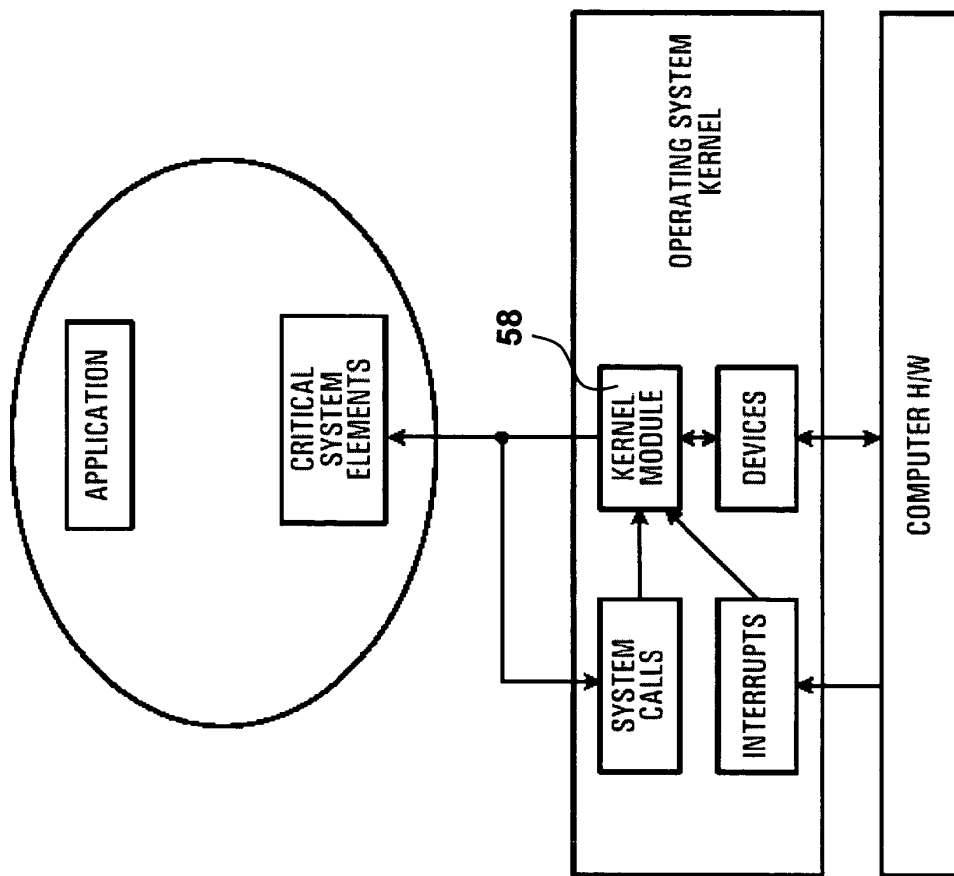


FIG. 5

U.S. Patent

Aug. 24, 2010

Sheet 7 of 7

US 7,784,058 B2

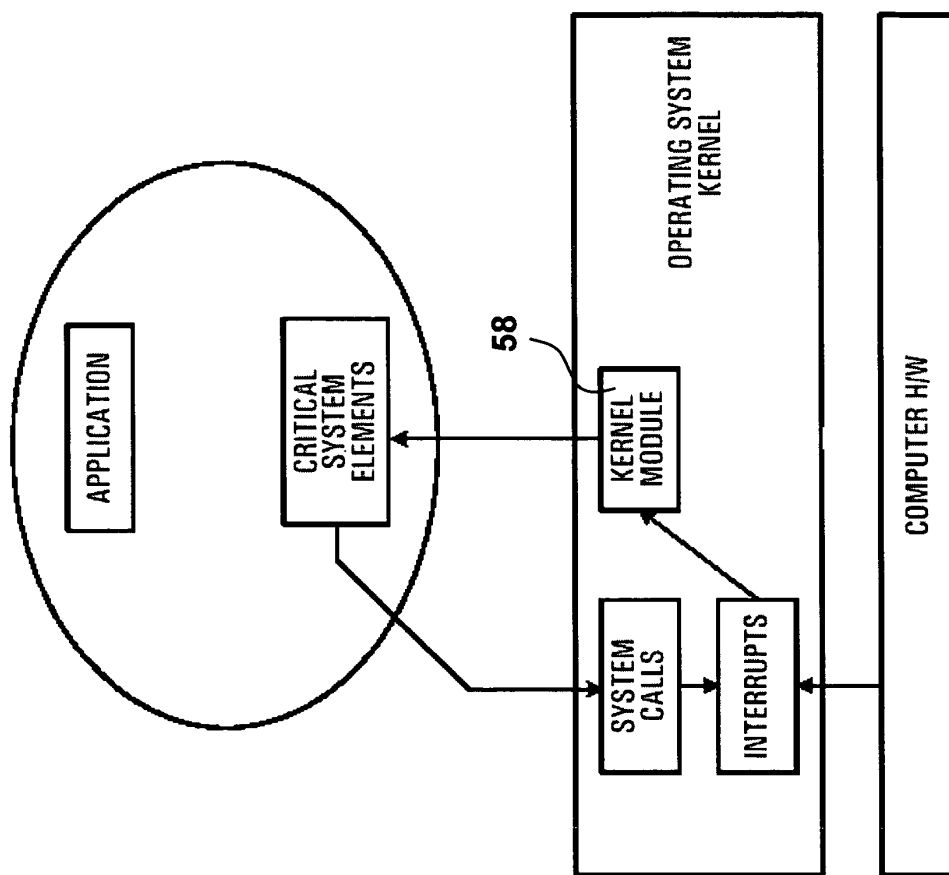


FIG. 6

US 7,784,058 B2

1

**COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled “User Mode Critical System Element as Shared Libs”, which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

2

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.

2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

US 7,784,058 B2

3

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to inter-cept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

4

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. 1 is an architectural view of the traditional monolithic prior art operating system;

US 7,784,058 B2

5

FIG. 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. 3 is an architectural view of an embodiment of the invention;

FIG. 4 is a functional view showing how critical system elements exist in the same context as an application;

FIG. 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. 6 shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TCP/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP

6

stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality

2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided

2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b

US 7,784,058 B2

7

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12a**, **12b** makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12a**, **12b** in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2a** shows a system architecture where critical system elements **27a**, **27b** execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2a** is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2a** as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2b** is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2a** and FIG. **2b** is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2b** is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the

8

case with a multi-server architecture depicted in FIG. **2a**. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32a**, **32b**, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45,47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

US 7,784,058 B2

9

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. 4. FIG. 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

10

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:

- a) a processor;
- b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,
- c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

US 7,784,058 B2

11

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

6. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

7. A computing system according to claim 6 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

8. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

12

9. A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

10. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

11. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

12. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

13. A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

14. A computing system according to claim 13 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

15. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

16. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

17. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.

* * * * *

Exhibit 4

U.S. Patent No. 7,784,058 vs. Microsoft

Accused Instrumentalities: Microsoft products and services using user mode critical system elements as shared libraries, including without limitation Azure Kubernetes Service (“AKS”), Azure Arc-enabled Kubernetes, Azure Container Registry, and Azure Container Apps, and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
[1pre] 1. A computing system for executing a plurality of software applications comprising:	<p>To the extent the preamble is limiting, each Accused Instrumentality comprises or constitutes a computing system for executing a plurality of software applications as claimed.</p> <p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <p>Azure Kubernetes Service (AKS) is a managed Kubernetes service that you can use to deploy and manage containerized applications. You need minimal container orchestration expertise to use AKS. AKS reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure. AKS is an ideal platform for deploying and managing containerized applications that require high availability, scalability, and portability, and for deploying applications to multiple regions, using open-source tools, and integrating with existing DevOps tools.</p> <p>https://learn.microsoft.com/en-us/azure/aks/what-is-aks</p>

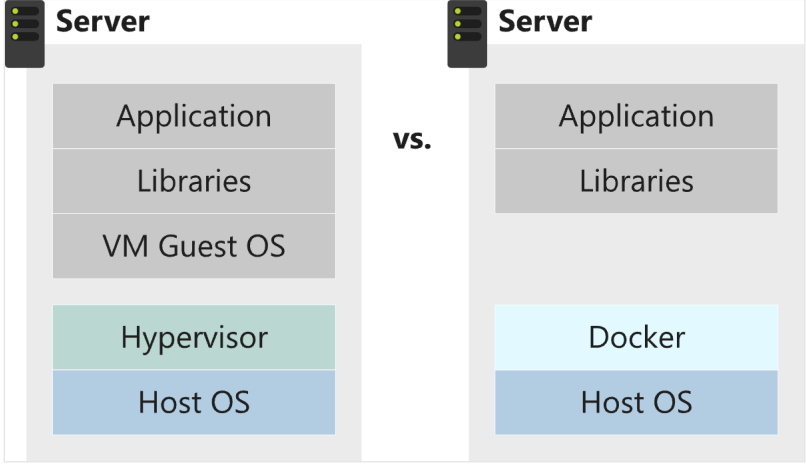
Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 266 1058 310">When to use AKS</h2> <p data-bbox="730 347 1377 375">The following list describes some common use cases for AKS:</p> <ul data-bbox="764 410 1911 935" style="list-style-type: none"><li data-bbox="764 410 1911 477">• Lift and shift to containers with AKS: Migrate existing applications to containers and run them in a fully managed Kubernetes environment.<li data-bbox="764 485 1911 552">• Microservices with AKS: Simplify the deployment and management of microservices-based applications with streamlined horizontal scaling, self-healing, load balancing, and secret management.<li data-bbox="764 560 1911 626">• Secure DevOps for AKS: Efficiently balance speed and security by implementing secure DevOps with Kubernetes.<li data-bbox="764 634 1911 701">• Bursting from AKS with ACI: Use virtual nodes to provision pods inside ACI that start in seconds and scale to meet demand.<li data-bbox="764 709 1911 776">• Machine learning model training with AKS: Train models using large datasets with familiar tools, such as TensorFlow and Kubeflow.<li data-bbox="764 784 1911 850">• Data streaming with AKS: Ingest and process real-time data streams with millions of data points collected via sensors, and perform fast analyses and computations to develop insights into complex scenarios.<li data-bbox="764 859 1911 925">• Using Windows containers on AKS: Run Windows Server containers on AKS to modernize your Windows applications and infrastructure. <p data-bbox="709 967 1377 995">https://learn.microsoft.com/en-us/azure/aks/what-is-aks</p>

Claim 1	Accused Instrumentalities
	<p>Azure Arc-enabled Kubernetes allows you to attach Kubernetes clusters running anywhere so that you can manage and configure them in Azure. By managing all of your Kubernetes resources in a single control plane, you can enable a more consistent development and operation experience, helping you run cloud-native apps anywhere and on any Kubernetes platform.</p> <p>When the Azure Arc agents are deployed to the cluster, an outbound connection to Azure is initiated, using industry-standard SSL to secure data in transit.</p> <p>Clusters that you connect to Azure are represented as their own resources in Azure Resource Manager, and they can be organized using resource groups and tagging.</p> <p>https://learn.microsoft.com/en-us/azure/azure-arc/kubernetes/overview</p> <p>Containers are becoming the preferred way to package, deploy, and manage cloud applications. Azure Container Instances offers the fastest and simplest way to run Linux or Windows containers in Azure, without having to manage any virtual machines and without having to adopt a higher-level service.</p> <p>ACI supports regular, confidential, and Spot containers. ACI can be used as single-instance or multi-instance via NGroups, or you can get orchestration capabilities by deploying pods in your Azure Kubernetes Service (AKS) cluster via virtual nodes on ACI. For even faster startup times, ACI supports standby pools.</p> <p>https://learn.microsoft.com/en-us/azure/container-instances/container-instances-overview</p>

Claim 1	Accused Instrumentalities
	<p>Azure Container Apps is a serverless platform that allows you to maintain less infrastructure and save costs while running containerized applications. Instead of worrying about server configuration, container orchestration, and deployment details, Container Apps provides all the up-to-date server resources required to keep your applications stable and secure.</p> <p>Common uses of Azure Container Apps include:</p> <ul style="list-style-type: none">• Deploying API endpoints• Hosting background processing jobs• Handling event-driven processing• Running microservices <p>https://learn.microsoft.com/en-us/azure/container-apps/overview</p> <h2>What is Kubernetes?</h2> <p>Kubernetes is an open-source container orchestration platform for automating the deployment, scaling, and management of containerized applications. For more information, see the official Kubernetes documentation.</p> <h2>What is AKS?</h2> <p>AKS is a managed Kubernetes service that simplifies deploying, managing, and scaling containerized applications using Kubernetes. For more information, see What is Azure Kubernetes Service (AKS)?</p> <p>https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts</p>

Claim 1	Accused Instrumentalities
<p>[1a] a) a processor;</p>	<p>Each Accused Instrumentality comprises a processor.</p> <p>For example, each node/host contains at least one CPU.</p> <p><i>See, e.g.:</i></p> <h2 data-bbox="726 443 1297 508">Node configuration</h2> <h2 data-bbox="726 581 1180 646">VM size and image</h2> <p>The Azure VM size for your nodes defines CPUs, memory, size, and the storage type available, such as high-performance SSD or regular HDD. The VM size you choose depends on the workload requirements and the number of pods you plan to run on each node. For more information, see Supported VM sizes in Azure Kubernetes Service (AKS).</p> <p>In AKS, the VM image for your cluster's nodes is based on Ubuntu Linux, Azure Linux, or Windows Server 2022. When you create an AKS cluster or scale out the number of nodes, the Azure platform automatically creates and configures the requested number of VMs. Agent nodes are billed as standard VMs, so any VM size discounts, including Azure reservations, are automatically applied.</p> <p>https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts</p>

Claim 1	Accused Instrumentalities
<p>[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,</p>	<p>Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.</p> <p>For example, the OSCSEs include kernel-mode functions similar to the functionalities provided by user-space libraries such as glibc. These are implemented in kernel-space to handle tasks such as (without limitation) memory management (kmalloc(), kfree(), etc.) at kernel level.</p> <p><i>See, e.g.:</i></p> <h2 data-bbox="743 537 1190 586">Container runtime</h2> <p>A container runtime is software that executes containers and manages container images on a node. The runtime helps abstract away sys-calls or OS-specific functionality to run containers on Linux or Windows. For Linux node pools, containerd is used on Kubernetes version 1.19 and higher. For Windows Server 2019 and 2022 node pools, containerd is generally available and is the only runtime option on Kubernetes version 1.23 and higher.</p> <p>https://learn.microsoft.com/en-us/azure/aks/core-aks-concepts</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="722 264 1026 305">Docker benefits</h2> <p data-bbox="722 337 1533 362">When we use Docker, we immediately get access to the benefits containerization offer.</p> <h2 data-bbox="722 410 1075 451">Efficient hardware use</h2> <p data-bbox="722 483 1915 540">Containers run without using a virtual machine (VM). As we learned, the container relies on the host kernel for functions such as file system, network management, process scheduling, and memory management.</p> <div data-bbox="722 565 1522 1027"><p>The diagram illustrates the architectural differences between a Virtual Machine (VM) and Docker containers. On the left, the VM architecture is shown as a stack of five layers: 'Application' (top), 'Libraries', 'VM Guest OS', 'Hypervisor', and 'Host OS' (bottom). On the right, the Docker architecture is shown as a stack of four layers: 'Application' (top), 'Libraries', 'Docker', and 'Host OS' (bottom). A 'vs.' label is placed between the two stacks, highlighting the removal of the 'VM Guest OS' and 'Hypervisor' layers in the Docker model.</p></div> <p data-bbox="722 1060 1915 1157">Compared to a VM, we can see that a VM requires an OS installed to provide kernel functions to the running applications inside the VM. Keep in mind that the VM OS also requires disk space, memory, and CPU time. By removing the VM and the additional OS requirement, we can free resources on the host and use it for running other containers.</p> <p data-bbox="722 1174 1890 1239">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

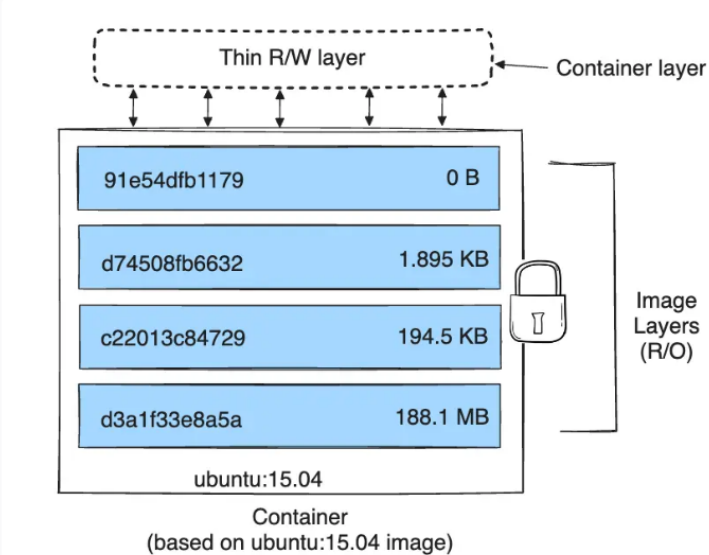
Claim 1	Accused Instrumentalities
	<p>Kernel mode</p> <p>Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p>https://www.techtarget.com/searchdatacenter/definition/kernel</p> <p>The GNU C Library, commonly known as glibc, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system.</p> <p>https://en.wikipedia.org/wiki/Glibc</p>

Claim 1	Accused Instrumentalities
<p>[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and</p>	<p>Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.</p> <p>For example, the shared library with SLCSEs include the runtime environment, system tools, and dependencies, such as the glibc library and other libraries that replicate OSCSEs, included in the container image (including without limitation in a base image that is included within the container image).</p> <p><i>See, e.g.:</i></p> <p>A container virtualizes the underlying OS and causes the containerized app to perceive that it has the OS—including CPU, memory, file storage, and network connections—all to itself. Because the differences in underlying OS and infrastructure are abstracted, as long as the base image is consistent, the container can be deployed and run anywhere. For developers, this is incredibly attractive.</p> <p>https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-container/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="726 264 1398 329">What are base images?</h2> <p data-bbox="726 375 1892 719">Dockerfiles defining most container images specify a parent image from which the image is based, often referred to as its <i>base image</i>. Base images typically contain the operating system, for example Alpine Linux or Windows Nano Server, on which the rest of the container's layers are applied. They might also include application frameworks such as Node.js or .NET Core. These base images are themselves typically based on public upstream images. Several of your application images might share a common base image.</p> <p data-bbox="714 751 1850 781">https://learn.microsoft.com/en-us/azure/container-registry/container-registry-tasks-base-images</p> <p data-bbox="732 826 1913 1070">In some cases, such as a private development team, a base image might specify more than OS or framework. For example, a base image could be a shared service component image that needs to be tracked. Members of a team might need to track this base image for testing, or need to regularly update the image when developing application images.</p> <p data-bbox="714 1083 1850 1112">https://learn.microsoft.com/en-us/azure/container-registry/container-registry-tasks-base-images</p> <h2 data-bbox="747 1144 1087 1190">Container images</h2> <p data-bbox="747 1214 1398 1330">A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p data-bbox="714 1349 1272 1378">https://kubernetes.io/docs/concepts/containers/</p>

Claim 1	Accused Instrumentalities
	<p>Container image files are complete, static and executable versions of an application or service and differ from one technology to another. Docker images are made up of multiple layers, which start with a base image that includes all of the dependencies needed to execute code in a container. Each image has a readable/writable layer on top of static unchanging layers. Because each container has its own specific container layer that customizes that specific container, underlying image layers can be saved and reused in multiple containers. An Open Container Initiative (OCI)</p> <p>https://www.techtarget.com/searchitoperations/definition/container-containerization-or-container-based-virtualization</p> <h2>About storage drivers</h2> <p>To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2>Storage drivers versus Docker volumes</h2> <p>Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p>Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="735 267 1129 316">Images and layers</h2> <p data-bbox="735 354 1827 423">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="735 467 1480 803"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="735 846 1900 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="735 1154 1270 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer structure of a container. At the bottom, a box labeled 'Container (based on ubuntu:15.04 image)' contains a stack of four 'Image Layers (R/O)'. Each layer is represented by a blue rectangle with a hash and a size: '91e54dfb1179' (0 B), 'd74508fb6632' (1.895 KB), 'c22013c84729' (194.5 KB), and 'd3a1f33e8a5a' (188.1 MB). A padlock icon is shown next to the stack, indicating they are read-only. Above this stack is a dashed box labeled 'Thin R/W layer', which is identified as the 'Container layer' by an arrow. Bidirectional arrows connect the container layer to the top of the image layers stack.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 277 980 337">Volumes</h2> <p data-bbox="730 388 1902 509">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="714 531 1344 561">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="730 609 1266 654">Container environment</h2> <p data-bbox="730 690 1499 751">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="770 787 1476 938" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="714 971 1549 1002">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="732 272 940 331">Images</h2> <p data-bbox="732 362 1545 505">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="732 537 1551 607">You typically create a container image of your application and push it to a registry before referring to it in a <code>Pod</code>.</p> <p data-bbox="714 633 1365 662">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="728 703 980 756">Volumes</h2> <p data-bbox="728 794 1554 1209">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <code>volume</code> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="714 1235 1346 1265">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="747 277 1335 331"><h2>Open Container Initiative</h2></div> <div data-bbox="747 391 1226 435"><h3>Image Format Specification</h3></div> <div data-bbox="747 479 1890 548"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="747 581 1900 651"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="711 677 1503 743"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>





Claim 1	Accused Instrumentalities
	<div data-bbox="730 272 894 310">Overview</div> <div data-bbox="730 362 1892 586"><p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p></div> <div data-bbox="730 626 1906 980"><p>The diagram illustrates the components of an image manifest. On the left, a code block shows a Java class: <pre>public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } }</pre>. An arrow points from this code to a cylinder labeled 'layer' containing the paths <code>/bin/java</code>, <code>/opt/app.jar</code>, and <code>/lib/libc</code>. This cylinder is followed by a plus sign and a document icon labeled 'image index' containing a JSON snippet: <pre>{ "manifests": { "platform": { "os": "linux", ... } }</pre>. This is followed by another plus sign and a document icon labeled 'config' containing a JSON snippet: <pre>{ ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } }</pre></p></div> <div data-bbox="714 1008 1503 1073"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 1	Accused Instrumentalities
	<h2>OCI Image Configuration</h2> <p>An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p>This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="737 272 821 306">Layer</p> <ul data-bbox="764 342 1913 667" style="list-style-type: none">• Image filesystems are composed of <i>layers</i>.• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="737 716 924 750">Image JSON</p> <ul data-bbox="764 786 1913 1110" style="list-style-type: none">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.• This JSON is considered to be immutable, because changing it would change the computed ImageID.• Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="711 1138 1526 1198">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

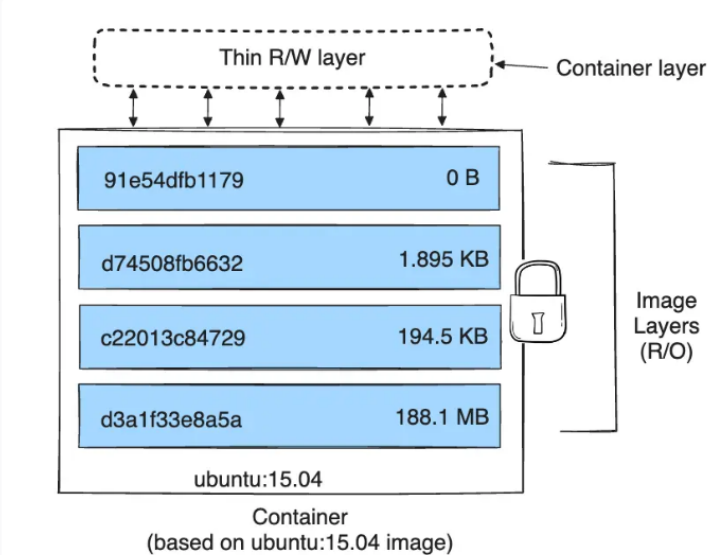
Claim 1	Accused Instrumentalities
	<p>The GNU C Library, commonly known as glibc, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system.</p> <p>https://en.wikipedia.org/wiki/Glibc</p>

Claim 1	Accused Instrumentalities
<p>[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,</p>	<p>In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.</p> <p>For example, a base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and dependencies, including system libraries or shared library critical system elements (SLCSEs). The base image forms a part of the container image according to the “layer” model described in the documentation below. When the container runs the image, it creates a runtime instance of that container image. In turn, when one or more applications executes within the container runtime environment, it dynamically links to the SLCSEs stored in the runtime environment, which thereby become a part of the application(s).</p> <p><i>See, e.g.:</i></p> <p>Container images</p> <p>A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p>https://kubernetes.io/docs/concepts/containers/</p>

Claim 1	Accused Instrumentalities																																																						
	<div>  <div> ubuntu  </div> <div> Updated 15 days ago Ubuntu is a Debian-based Linux operating system based on free software. </div> <div> Linux IBM Z 386 riscv64 x86-64 ARM ARM 64 PowerPC 64 LE </div> </div> <div> <div>  <div> debian  </div> <div> Updated 35 minutes ago Debian is a Linux distribution that's composed entirely of free and open-source software. </div> <div> Linux riscv64 x86-64 ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z </div> </div> <p>https://hub.docker.com/search?image_filter=official&type=image&q=</p> <table border="1"> <thead> <tr> <th>Platform</th> <th>x86_64 / amd64</th> <th>arm64 / aarch64</th> <th>arm (32-bit)</th> <th>ppc64le</th> <th>s390x</th> </tr> </thead> <tbody> <tr> <td>CentOS</td> <td>✓</td> <td>✓</td> <td></td> <td>✓</td> <td></td> </tr> <tr> <td>Debian</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td></td> </tr> <tr> <td>Fedora</td> <td>✓</td> <td>✓</td> <td></td> <td>✓</td> <td></td> </tr> <tr> <td>Raspberry Pi OS (32-bit)</td> <td></td> <td></td> <td>✓</td> <td></td> <td></td> </tr> <tr> <td>RHEL (s390x)</td> <td></td> <td></td> <td></td> <td></td> <td>✓</td> </tr> <tr> <td>SLES</td> <td></td> <td></td> <td></td> <td></td> <td>✓</td> </tr> <tr> <td>Ubuntu</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> </tr> <tr> <td>Binaries</td> <td>✓</td> <td>✓</td> <td>✓</td> <td></td> <td></td> </tr> </tbody> </table> <p>https://docs.docker.com/engine/install/</p> </div>	Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x	CentOS	✓	✓		✓		Debian	✓	✓	✓	✓		Fedora	✓	✓		✓		Raspberry Pi OS (32-bit)			✓			RHEL (s390x)					✓	SLES					✓	Ubuntu	✓	✓	✓	✓	✓	Binaries	✓	✓	✓		
Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x																																																		
CentOS	✓	✓		✓																																																			
Debian	✓	✓	✓	✓																																																			
Fedora	✓	✓		✓																																																			
Raspberry Pi OS (32-bit)			✓																																																				
RHEL (s390x)					✓																																																		
SLES					✓																																																		
Ubuntu	✓	✓	✓	✓	✓																																																		
Binaries	✓	✓	✓																																																				

Claim 1	Accused Instrumentalities
	<p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <h2>About storage drivers</h2> <p>To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2>Storage drivers versus Docker volumes</h2> <p>Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p>Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="735 267 1129 316">Images and layers</h2> <p data-bbox="735 354 1827 423">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="735 467 1480 803"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="735 846 1900 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="735 1154 1270 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer architecture of a container. At the bottom is a stack of four blue rectangular blocks representing 'Image Layers (R/O)'. From top to bottom, they are labeled with their IDs and sizes: '91e54dfb1179' (0 B), 'd74508fb6632' (1.895 KB), 'c22013c84729' (194.5 KB), and 'd3a1f33e8a5a' (188.1 MB). A bracket on the right side of these blocks is labeled 'Image Layers (R/O)'. Above this stack is a dashed-line box labeled 'Thin R/W layer'. An arrow points from the text 'Container layer' to this dashed box. Below the image layers, the text 'ubuntu:15.04' is centered. At the very bottom, the text 'Container (based on ubuntu:15.04 image)' is centered. A padlock icon is positioned to the right of the image layers, indicating they are read-only.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 277 982 337">Volumes</h2> <p data-bbox="730 386 1906 508">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="714 532 1344 561">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="730 607 1268 654">Container environment</h2> <p data-bbox="730 688 1501 751">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="772 786 1478 938" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="714 971 1551 1000">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="732 272 940 332">Images</h2> <p data-bbox="732 362 1545 505">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="732 540 1551 607">You typically create a container image of your application and push it to a registry before referring to it in a <code>Pod</code>.</p> <p data-bbox="714 633 1365 662">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="728 703 980 760">Volumes</h2> <p data-bbox="728 795 1554 1209">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <code>volume</code> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="714 1235 1346 1265">https://kubernetes.io/docs/concepts/storage/volumes/</p>

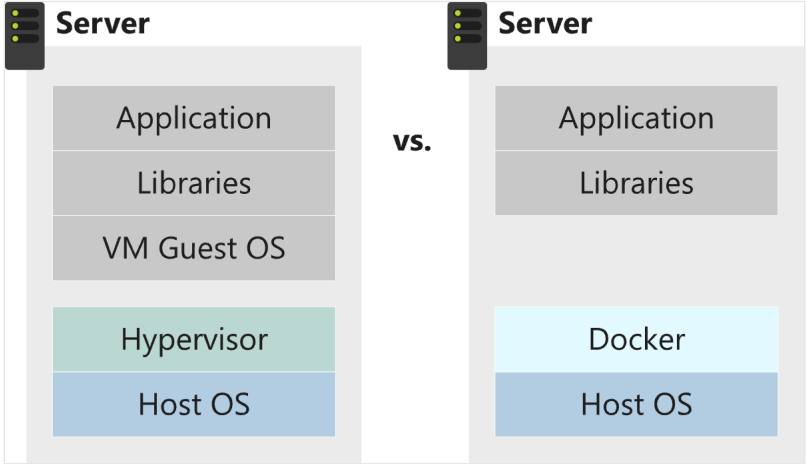
Claim 1	Accused Instrumentalities
	<div data-bbox="747 277 1335 331"><h2>Open Container Initiative</h2></div> <div data-bbox="747 391 1226 435"><h3>Image Format Specification</h3></div> <div data-bbox="747 479 1890 548"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="747 581 1900 651"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="711 677 1505 743"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 272 894 310">Overview</p> <p data-bbox="730 362 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="743 623 1906 980"> <p>The diagram illustrates the components of an image manifest. On the left, a code block for a Java class is shown: <code>public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } }</code>. An arrow points from this code to a cylinder labeled 'layer' containing the paths <code>/bin/java</code>, <code>/opt/app.jar</code>, and <code>/lib/libc</code>. This cylinder is followed by a plus sign and a document icon labeled 'image index' containing a JSON snippet: <code>{ "manifests": { "platform": { "os": "linux", ... } } }</code>. Another plus sign and document icon labeled 'config' follows, containing a JSON snippet: <code>{ ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } }</code>.</p> <p data-bbox="714 1008 1503 1073">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p> </div>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 266 1335 321">OCI Image Configuration</h2> <p data-bbox="730 370 1913 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="730 558 1671 589">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="714 621 1526 686">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

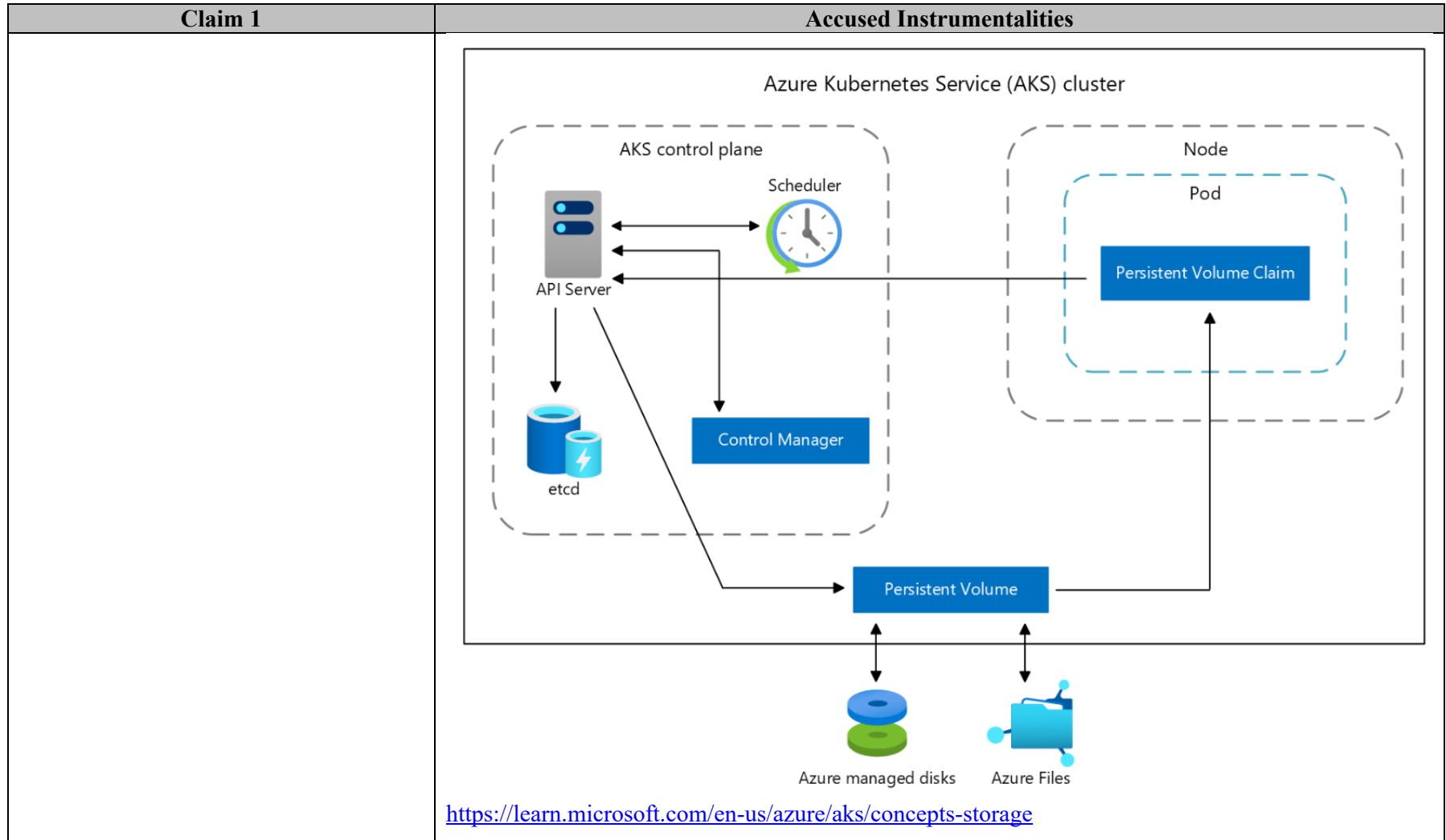
Claim 1	Accused Instrumentalities
	<p data-bbox="737 272 821 305">Layer</p> <ul data-bbox="764 342 1913 667" style="list-style-type: none">• Image filesystems are composed of <i>layers</i>.• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="737 716 924 748">Image JSON</p> <ul data-bbox="764 786 1913 1110" style="list-style-type: none">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.• This JSON is considered to be immutable, because changing it would change the computed ImageID.• Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="711 1138 1528 1198">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p> <p data-bbox="711 1226 1306 1291">Containers only have access to resources that are defined in the image, https://www.hpe.com/us/en/what-is/docker.html</p>

Claim 1	Accused Instrumentalities
	<p>DESCRIPTION top</p> <p>The programs ld.so and ld-linux.so* find and load the shared objects (shared libraries) needed by a program, prepare the program to run, and then run it.</p> <p>https://man7.org/linux/man-pages/man8/ld.so.8.html</p>
<p>[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and</p>	<p>In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.</p> <p>When a Docker or Kubernetes image is used to create a container in the Accused Instrumentalities, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications running in their respective containers. The image includes essential system files, libraries, and dependencies required to run the software application within the container. The containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it is not shared with other containers. This ensures that each container has its own isolated context. Docker or Kubernetes containers in the Accused Instrumentalities can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same function.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<div data-bbox="722 264 1029 305"><h2>Docker benefits</h2></div> <div data-bbox="722 337 1533 362"><p>When we use Docker, we immediately get access to the benefits containerization offer.</p></div> <div data-bbox="722 410 1075 446"><h2>Efficient hardware use</h2></div> <div data-bbox="722 482 1915 539"><p>Containers run without using a virtual machine (VM). As we learned, the container relies on the host kernel for functions such as file system, network management, process scheduling, and memory management.</p></div> <div data-bbox="722 563 1522 1024"><p>The diagram illustrates two server architectures side-by-side, separated by a 'vs.' label. The left server, labeled 'Server', shows a stack of layers: 'Application' (grey), 'Libraries' (grey), 'VM Guest OS' (grey), 'Hypervisor' (teal), and 'Host OS' (blue). The right server, also labeled 'Server', shows a similar stack: 'Application' (grey), 'Libraries' (grey), 'Docker' (light blue), and 'Host OS' (blue). The 'Docker' layer is positioned where the 'VM Guest OS' and 'Hypervisor' layers would be in the VM architecture, demonstrating a more streamlined stack.</p></div> <div data-bbox="722 1060 1915 1153"><p>Compared to a VM, we can see that a VM requires an OS installed to provide kernel functions to the running applications inside the VM. Keep in mind that the VM OS also requires disk space, memory, and CPU time. By removing the VM and the additional OS requirement, we can free resources on the host and use it for running other containers.</p></div> <div data-bbox="714 1175 1890 1239"><p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p></div>




Claim 1	Accused Instrumentalities
	<p data-bbox="730 261 1031 293">Container isolation</p> <p data-bbox="730 329 1915 422">Docker containers provide security features to run multiple containers simultaneously on the same host without affecting each other. As we learned, we can configure both data storage and network configuration to isolate our containers or share data and connectivity between specific containers.</p> <p data-bbox="730 451 1102 475">Let's compare this feature to using VMs.</p> <div data-bbox="730 500 1524 873"> </div> <p data-bbox="730 906 1915 998">Assume we have a physical host running two VMs. We have three applications that we want to run isolated from each other. We decide to deploy the first app onto VM1 and the second onto VM2 to separate the two apps from each other. If we now choose to install the third application, we'll need to install another VM to continue this pattern.</p> <p data-bbox="714 1015 1890 1079">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="730 1112 1081 1144">Application portability</p> <p data-bbox="730 1180 1915 1239">Containers run almost everywhere: desktops, physical servers, VMs, and in the cloud. This runtime compatibility makes it easy to move containerized applications among different environments.</p> <p data-bbox="714 1255 1890 1320">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

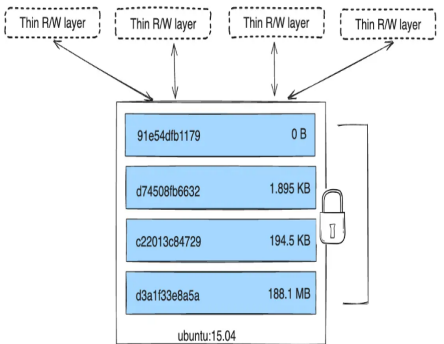
Claim 1	Accused Instrumentalities
	<p data-bbox="730 269 1037 310">Cloud deployments</p> <p data-bbox="730 337 1919 399">Docker containers are the default container architecture the Azure containerization services use, and many other cloud platforms also support them.</p> <p data-bbox="730 427 1919 488">For instance, you can deploy Docker containers to Azure Container Instances, Azure App Service, and Azure Kubernetes Services. Each of these options provides you with different features and capabilities.</p> <p data-bbox="730 516 1919 610">For example, Azure container instances allow you to focus on designing and building your applications without the overhead of managing infrastructure. When you have many containers to orchestrate, Azure Kubernetes service makes it easy to deploy and manage large-scale container deployments.</p> <p data-bbox="714 630 1892 691">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p data-bbox="730 743 1801 1146">A container virtualizes the underlying OS and causes the containerized app to perceive that it has the OS—including CPU, memory, file storage, and network connections—all to itself. Because the differences in underlying OS and infrastructure are abstracted, as long as the base image is consistent, the container can be deployed and run anywhere. For developers, this is incredibly attractive.</p> <p data-bbox="714 1182 1829 1211">https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-container/</p>

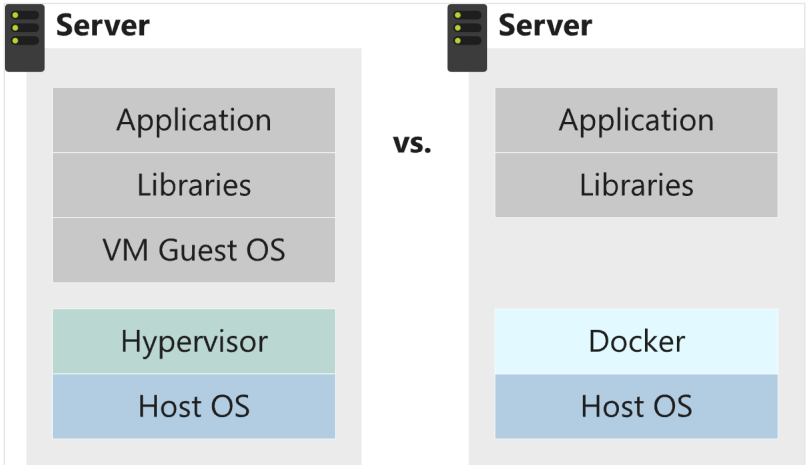


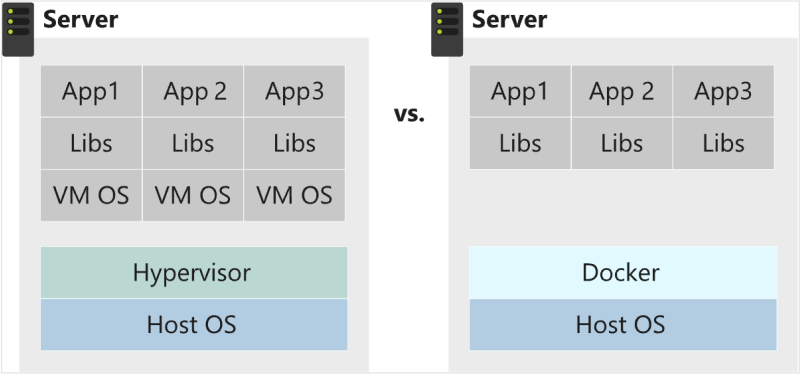
Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 1163 326">Ephemeral OS disk</h2> <p data-bbox="737 358 1913 500">By default, Azure automatically replicates the operating system disk for a virtual machine to Azure Storage to avoid data loss when the VM is relocated to another host. However, since containers aren't designed to have local state persisted, this behavior offers limited value while providing some drawbacks. These drawbacks include, but aren't limited to, slower node provisioning and higher read/write latency.</p> <p data-bbox="737 532 1822 597">By contrast, ephemeral OS disks are stored only on the host machine, just like a temporary disk. With this configuration, you get lower read/write latency, together with faster node scaling and cluster upgrades.</p> <p data-bbox="711 613 1440 643">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p> <h2 data-bbox="737 688 930 742">Volumes</h2> <p data-bbox="737 774 1860 873">Kubernetes typically treats individual pods as ephemeral, disposable resources. Applications have different approaches available to them for using and persisting data. A <i>volume</i> represents a way to store, retrieve, and persist data across pods and through the application lifecycle.</p> <p data-bbox="737 906 1898 1011">Traditional volumes are created as Kubernetes resources backed by Azure Storage. You can manually create data volumes to be assigned to pods directly or have Kubernetes automatically create them. Data volumes can use: Azure Disk, Azure Files, Azure NetApp Files, or Azure Blobs.</p> <p data-bbox="711 1027 1440 1057">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 264 1104 305">Persistent volumes</h2> <p data-bbox="730 337 1764 462">Volumes defined and created as part of the pod lifecycle only exist until you delete the pod. Pods often expect their storage to remain if a pod is rescheduled on a different host during a maintenance event, especially in StatefulSets. A <i>persistent volume</i> (PV) is a storage resource created and managed by the Kubernetes API that can exist beyond the lifetime of an individual pod.</p> <p data-bbox="730 495 1491 516">You can use the following Azure Storage services to provide the persistent volume:</p> <ul data-bbox="756 548 1003 641" style="list-style-type: none">• Azure Disk• Azure Files• Azure Container Storage <p data-bbox="730 673 1732 727">As noted in the Volumes section, the choice of Azure Disks or Azure Files is often determined by the need for concurrent access to the data or the performance tier.</p> <div data-bbox="739 760 1768 1295"><p>The diagram illustrates the connection between an Azure Kubernetes Service (AKS) cluster and Azure storage services. A rounded rectangle labeled 'Azure Kubernetes Service (AKS) cluster' contains a blue box labeled 'Persistent Volume'. Two arrows originate from this 'Persistent Volume' box. One arrow points to a stack of two disks (one blue, one green) labeled 'Azure managed disks (Standard or Premium storage)', with the text 'Single node/pod access' above it. The other arrow points to a blue folder icon labeled 'Azure Files (Standard storage)', with the text 'Multiple concurrent node/pod access' above it.</p></div> <p data-bbox="714 1328 1440 1356">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p>

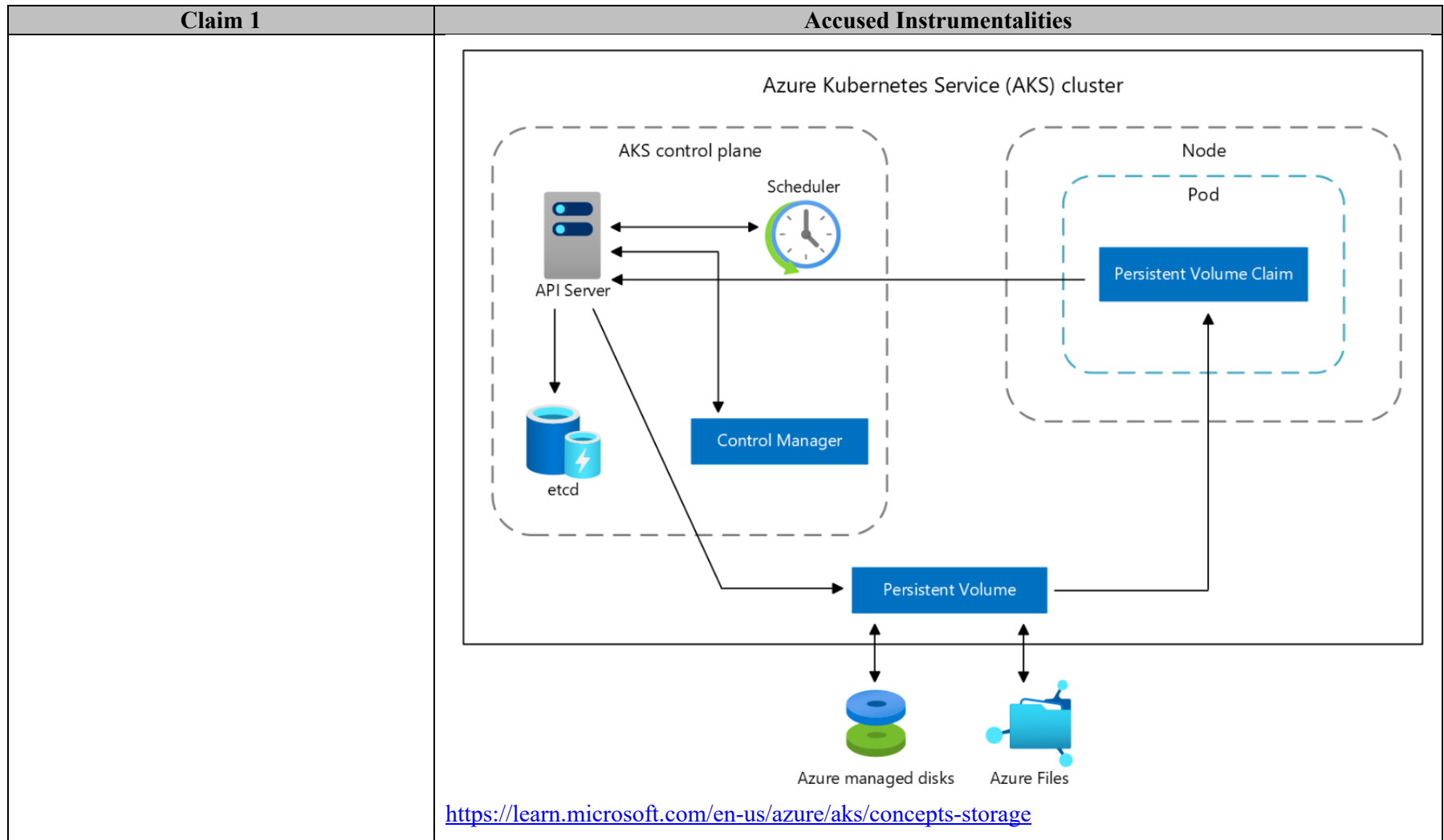
Claim 1	Accused Instrumentalities
	<p data-bbox="726 272 898 310">Overview</p> <p data-bbox="726 362 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="743 623 1906 980"> <pre data-bbox="743 753 1037 841"> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } } </pre> <p data-bbox="1058 786 1100 802">→</p> <div data-bbox="1121 623 1906 980"> <div data-bbox="1121 623 1346 943">  <div data-bbox="1163 753 1325 841"> /bin/java /opt/app.jar /lib/libc </div> </div> <div data-bbox="1367 786 1398 818">+</div> <div data-bbox="1398 623 1633 943">  <div data-bbox="1419 704 1612 850"> { "manifests": { "platform": { "os": "linux", ... } } } </div> </div> <div data-bbox="1640 786 1671 818">+</div> <div data-bbox="1671 623 1906 943">  <div data-bbox="1692 672 1885 883"> { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } } </div> </div> </div> <div data-bbox="1205 948 1268 980">layer</div> <div data-bbox="1457 948 1583 980">image index</div> <div data-bbox="1772 948 1835 980">config</div> </div> <p data-bbox="709 1008 1503 1073"> https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md </p>

Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p>
<p>[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p>	<p>In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p> <p>For example, in Docker or Kubernetes containers in the Accused Instrumentalities, each container operates independently, and a base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When an image is used to create a container in the Accused Instrumentality, an instance of the SLCSE is provided to a software</p>

Claim 1	Accused Instrumentalities
	<p>application. Therefore, different instances of the SLCSE are provided to different applications for performing a same function, simultaneously.</p> <p><i>See, e.g.:</i></p> <h2 data-bbox="722 402 1026 446">Docker benefits</h2> <p data-bbox="722 479 1533 503">When we use Docker, we immediately get access to the benefits containerization offer.</p> <h2 data-bbox="722 552 1075 589">Efficient hardware use</h2> <p data-bbox="722 621 1915 682">Containers run without using a virtual machine (VM). As we learned, the container relies on the host kernel for functions such as file system, network management, process scheduling, and memory management.</p>  <p data-bbox="722 1201 1915 1295">Compared to a VM, we can see that a VM requires an OS installed to provide kernel functions to the running applications inside the VM. Keep in mind that the VM OS also requires disk space, memory, and CPU time. By removing the VM and the additional OS requirement, we can free resources on the host and use it for running other containers.</p> <p data-bbox="714 1315 1890 1380">https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p>

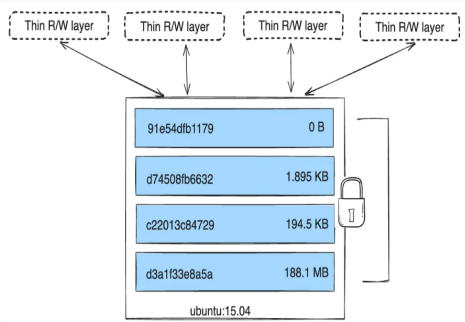
Claim 1	Accused Instrumentalities
	<div data-bbox="730 261 1031 297"><h3>Container isolation</h3></div> <div data-bbox="730 329 1915 422"><p>Docker containers provide security features to run multiple containers simultaneously on the same host without affecting each other. As we learned, we can configure both data storage and network configuration to isolate our containers or share data and connectivity between specific containers.</p></div> <div data-bbox="730 451 1102 477"><p>Let's compare this feature to using VMs.</p></div> <div data-bbox="730 500 1524 872"><p>The diagram compares two architectures for running applications. On the left, a 'Server' icon is shown above a stack of layers: 'Host OS' (blue), 'Hypervisor' (green), and three 'VM OS' (grey) boxes. Each 'VM OS' box contains 'App1', 'App2', and 'App3' (grey), and 'Libs' (grey). On the right, separated by 'vs.', is another 'Server' icon above a stack: 'Host OS' (blue), 'Docker' (light blue), and three application boxes. Each application box contains 'App1', 'App2', and 'App3' (grey), and 'Libs' (grey).</p></div> <div data-bbox="730 906 1915 998"><p>Assume we have a physical host running two VMs. We have three applications that we want to run isolated from each other. We decide to deploy the first app onto VM1 and the second onto VM2 to separate the two apps from each other. If we now choose to install the third application, we'll need to install another VM to continue this pattern.</p></div> <div data-bbox="730 1015 1915 1081"><p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p></div> <div data-bbox="730 1110 1081 1151"><h3>Application portability</h3></div> <div data-bbox="730 1180 1915 1240"><p>Containers run almost everywhere: desktops, physical servers, VMs, and in the cloud. This runtime compatibility makes it easy to move containerized applications among different environments.</p></div> <div data-bbox="730 1255 1915 1321"><p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p></div>

Claim 1	Accused Instrumentalities
	<p>Cloud deployments</p> <p>Docker containers are the default container architecture the Azure containerization services use, and many other cloud platforms also support them.</p> <p>For instance, you can deploy Docker containers to Azure Container Instances, Azure App Service, and Azure Kubernetes Services. Each of these options provides you with different features and capabilities.</p> <p>For example, Azure container instances allow you to focus on designing and building your applications without the overhead of managing infrastructure. When you have many containers to orchestrate, Azure Kubernetes service makes it easy to deploy and manage large-scale container deployments.</p> <p>https://learn.microsoft.com/en-us/training/modules/intro-to-docker-containers/5-when-use-docker-containers</p> <p>A container virtualizes the underlying OS and causes the containerized app to perceive that it has the OS—including CPU, memory, file storage, and network connections—all to itself. Because the differences in underlying OS and infrastructure are abstracted, as long as the base image is consistent, the container can be deployed and run anywhere. For developers, this is incredibly attractive.</p> <p>https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-container/</p>



Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 1163 326">Ephemeral OS disk</h2> <p data-bbox="737 358 1913 500">By default, Azure automatically replicates the operating system disk for a virtual machine to Azure Storage to avoid data loss when the VM is relocated to another host. However, since containers aren't designed to have local state persisted, this behavior offers limited value while providing some drawbacks. These drawbacks include, but aren't limited to, slower node provisioning and higher read/write latency.</p> <p data-bbox="737 532 1822 597">By contrast, ephemeral OS disks are stored only on the host machine, just like a temporary disk. With this configuration, you get lower read/write latency, together with faster node scaling and cluster upgrades.</p> <p data-bbox="711 613 1440 643">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p> <h2 data-bbox="737 686 930 740">Volumes</h2> <p data-bbox="737 773 1860 873">Kubernetes typically treats individual pods as ephemeral, disposable resources. Applications have different approaches available to them for using and persisting data. A <i>volume</i> represents a way to store, retrieve, and persist data across pods and through the application lifecycle.</p> <p data-bbox="737 906 1898 1011">Traditional volumes are created as Kubernetes resources backed by Azure Storage. You can manually create data volumes to be assigned to pods directly or have Kubernetes automatically create them. Data volumes can use: Azure Disk, Azure Files, Azure NetApp Files, or Azure Blobs.</p> <p data-bbox="711 1027 1440 1057">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 264 1104 305">Persistent volumes</h2> <p data-bbox="730 337 1764 464">Volumes defined and created as part of the pod lifecycle only exist until you delete the pod. Pods often expect their storage to remain if a pod is rescheduled on a different host during a maintenance event, especially in StatefulSets. A <i>persistent volume</i> (PV) is a storage resource created and managed by the Kubernetes API that can exist beyond the lifetime of an individual pod.</p> <p data-bbox="730 493 1491 516">You can use the following Azure Storage services to provide the persistent volume:</p> <ul data-bbox="756 548 1003 639" style="list-style-type: none">• Azure Disk• Azure Files• Azure Container Storage <p data-bbox="730 669 1732 727">As noted in the Volumes section, the choice of Azure Disks or Azure Files is often determined by the need for concurrent access to the data or the performance tier.</p> <div data-bbox="739 760 1768 1295"><p>The diagram illustrates the architecture of Persistent Volumes in an Azure Kubernetes Service (AKS) cluster. A central box labeled 'Azure Kubernetes Service (AKS) cluster' contains a blue box labeled 'Persistent Volume'. Two arrows originate from the 'Persistent Volume' box and point to external storage options. The top arrow points to a stack of blue and green disks labeled 'Azure managed disks (Standard or Premium storage)', with the text 'Single node/pod access' above it. The bottom arrow points to a blue folder icon labeled 'Azure Files (Standard storage)', with the text 'Multiple concurrent node/pod access' above it.</p></div> <p data-bbox="714 1328 1440 1357">https://learn.microsoft.com/en-us/azure/aks/concepts-storage</p>

Claim 1	Accused Instrumentalities
	<p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <p>A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.</p> <p>https://docs.docker.com/get-started/overview/</p> <p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>

JS 44 (Rev. 10/20)

CIVIL COVER SHEET

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. (SEE INSTRUCTIONS ON NEXT PAGE OF THIS FORM.)

I. (a) PLAINTIFFS

VIRTAMOVE, CORP.

(b) County of Residence of First Listed Plaintiff _____
(EXCEPT IN U.S. PLAINTIFF CASES)

(c) Attorneys (Firm Name, Address, and Telephone Number)

Russ August & Kabat, 12424 Wilshire Boulevard, 12th
Floor, Los Angeles, California 90025, Tel.: (310) 826-7474

DEFENDANTS

MICROSOFT CORPORATION

County of Residence of First Listed Defendant _____
(IN U.S. PLAINTIFF CASES ONLY)

NOTE: IN LAND CONDEMNATION CASES, USE THE LOCATION OF
THE TRACT OF LAND INVOLVED.

Attorneys (If Known)

II. BASIS OF JURISDICTION (Place an "X" in One Box Only)

- ☐ 1 U.S. Government Plaintiff ☒ 3 Federal Question
(U.S. Government Not a Party)
- ☐ 2 U.S. Government Defendant ☐ 4 Diversity
(Indicate Citizenship of Parties in Item III)

III. CITIZENSHIP OF PRINCIPAL PARTIES (Place an "X" in One Box for Plaintiff and One Box for Defendant)

- | | PTF | DEF | | PTF | DEF |
|---|----------------------------|----------------------------|---|----------------------------|----------------------------|
| Citizen of This State | <input type="checkbox"/> 1 | <input type="checkbox"/> 1 | Incorporated or Principal Place of Business In This State | <input type="checkbox"/> 4 | <input type="checkbox"/> 4 |
| Citizen of Another State | <input type="checkbox"/> 2 | <input type="checkbox"/> 2 | Incorporated and Principal Place of Business In Another State | <input type="checkbox"/> 5 | <input type="checkbox"/> 5 |
| Citizen or Subject of a Foreign Country | <input type="checkbox"/> 3 | <input type="checkbox"/> 3 | Foreign Nation | <input type="checkbox"/> 6 | <input type="checkbox"/> 6 |

IV. NATURE OF SUIT (Place an "X" in One Box Only)Click here for: [Nature of Suit Code Descriptions.](#)

CONTRACT	TORTS	FORFEITURE/PENALTY	BANKRUPTCY	OTHER STATUTES
<input type="checkbox"/> 110 Insurance <input type="checkbox"/> 120 Marine <input type="checkbox"/> 130 Miller Act <input type="checkbox"/> 140 Negotiable Instrument <input type="checkbox"/> 150 Recovery of Overpayment & Enforcement of Judgment <input type="checkbox"/> 151 Medicare Act <input type="checkbox"/> 152 Recovery of Defaulted Student Loans (Excludes Veterans) <input type="checkbox"/> 153 Recovery of Overpayment of Veteran's Benefits <input type="checkbox"/> 160 Stockholders' Suits <input type="checkbox"/> 190 Other Contract <input type="checkbox"/> 195 Contract Product Liability <input type="checkbox"/> 196 Franchise	PERSONAL INJURY <input type="checkbox"/> 310 Airplane <input type="checkbox"/> 315 Airplane Product Liability <input type="checkbox"/> 320 Assault, Libel & Slander <input type="checkbox"/> 330 Federal Employers' Liability <input type="checkbox"/> 340 Marine <input type="checkbox"/> 345 Marine Product Liability <input type="checkbox"/> 350 Motor Vehicle <input type="checkbox"/> 355 Motor Vehicle Product Liability <input type="checkbox"/> 360 Other Personal Injury <input type="checkbox"/> 362 Personal Injury - Medical Malpractice PERSONAL INJURY <input type="checkbox"/> 365 Personal Injury - Product Liability <input type="checkbox"/> 367 Health Care/Pharmaceutical Personal Injury Product Liability <input type="checkbox"/> 368 Asbestos Personal Injury Product Liability PERSONAL PROPERTY <input type="checkbox"/> 370 Other Fraud <input type="checkbox"/> 371 Truth in Lending <input type="checkbox"/> 380 Other Personal Property Damage <input type="checkbox"/> 385 Property Damage Product Liability	<input type="checkbox"/> 625 Drug Related Seizure of Property 21 USC 881 <input type="checkbox"/> 690 Other LABOR <input type="checkbox"/> 710 Fair Labor Standards Act <input type="checkbox"/> 720 Labor/Management Relations <input type="checkbox"/> 740 Railway Labor Act <input type="checkbox"/> 751 Family and Medical Leave Act <input type="checkbox"/> 790 Other Labor Litigation <input type="checkbox"/> 791 Employee Retirement Income Security Act IMMIGRATION <input type="checkbox"/> 462 Naturalization Application <input type="checkbox"/> 465 Other Immigration Actions	<input type="checkbox"/> 422 Appeal 28 USC 158 <input type="checkbox"/> 423 Withdrawal 28 USC 157 PROPERTY RIGHTS <input checked="" type="checkbox"/> 820 Copyrights <input type="checkbox"/> 830 Patent <input type="checkbox"/> 835 Patent - Abbreviated New Drug Application <input type="checkbox"/> 840 Trademark <input type="checkbox"/> 880 Defend Trade Secrets Act of 2016 SOCIAL SECURITY <input type="checkbox"/> 861 HIA (1395ff) <input type="checkbox"/> 862 Black Lung (923) <input type="checkbox"/> 863 DIWC/DIWW (405(g)) <input type="checkbox"/> 864 SSID Title XVI <input type="checkbox"/> 865 RSI (405(g)) FEDERAL TAX SUITS <input type="checkbox"/> 870 Taxes (U.S. Plaintiff or Defendant) <input type="checkbox"/> 871 IRS—Third Party 26 USC 7609	<input type="checkbox"/> 375 False Claims Act <input type="checkbox"/> 376 Qui Tam (31 USC 3729(a)) <input type="checkbox"/> 400 State Reapportionment <input type="checkbox"/> 410 Antitrust <input type="checkbox"/> 430 Banks and Banking <input type="checkbox"/> 450 Commerce <input type="checkbox"/> 460 Deportation <input type="checkbox"/> 470 Racketeer Influenced and Corrupt Organizations <input type="checkbox"/> 480 Consumer Credit (15 USC 1681 or 1692) <input type="checkbox"/> 485 Telephone Consumer Protection Act <input type="checkbox"/> 490 Cable/Sat TV <input type="checkbox"/> 850 Securities/Commodities/Exchange <input type="checkbox"/> 890 Other Statutory Actions <input type="checkbox"/> 891 Agricultural Acts <input type="checkbox"/> 893 Environmental Matters <input type="checkbox"/> 895 Freedom of Information Act <input type="checkbox"/> 896 Arbitration <input type="checkbox"/> 899 Administrative Procedure Act/Review or Appeal of Agency Decision <input type="checkbox"/> 950 Constitutionality of State Statutes
REAL PROPERTY <input type="checkbox"/> 210 Land Condemnation <input type="checkbox"/> 220 Foreclosure <input type="checkbox"/> 230 Rent Lease & Ejectment <input type="checkbox"/> 240 Torts to Land <input type="checkbox"/> 245 Tort Product Liability <input type="checkbox"/> 290 All Other Real Property	CIVIL RIGHTS <input type="checkbox"/> 440 Other Civil Rights <input type="checkbox"/> 441 Voting <input type="checkbox"/> 442 Employment <input type="checkbox"/> 443 Housing/Accommodations <input type="checkbox"/> 445 Amer. w/Disabilities - Employment <input type="checkbox"/> 446 Amer. w/Disabilities - Other <input type="checkbox"/> 448 Education PRISONER PETITIONS Habeas Corpus: <input type="checkbox"/> 463 Alien Detainee <input type="checkbox"/> 510 Motions to Vacate Sentence <input type="checkbox"/> 530 General <input type="checkbox"/> 535 Death Penalty Other: <input type="checkbox"/> 540 Mandamus & Other <input type="checkbox"/> 550 Civil Rights <input type="checkbox"/> 555 Prison Condition <input type="checkbox"/> 560 Civil Detainee - Conditions of Confinement			

V. ORIGIN (Place an "X" in One Box Only)

- ☒ 1 Original Proceeding ☐ 2 Removed from State Court ☐ 3 Remanded from Appellate Court ☐ 4 Reinstated or Reopened ☐ 5 Transferred from Another District (specify) ☐ 6 Multidistrict Litigation - Transfer ☐ 8 Multidistrict Litigation - Direct File

VI. CAUSE OF ACTION

Cite the U.S. Civil Statute under which you are filing (Do not cite jurisdictional statutes unless diversity):
35 U.S.C. § 1 et seq.

Brief description of cause:
Patent Infringement

VII. REQUESTED IN COMPLAINT:

☐ CHECK IF THIS IS A CLASS ACTION UNDER RULE 23, F.R.Cv.P.

DEMAND \$

CHECK YES only if demanded in complaint:

JURY DEMAND: ☒ Yes ☐ No**VIII. RELATED CASE(S) IF ANY**

(See instructions):

JUDGE Counts; Albright

DOCKET NUMBER 7:24-cv-00030; 7:24-cv-00033

DATE

Dec 20, 2024

SIGNATURE OF ATTORNEY OF RECORD

/s/ Reza Mirzaie

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG. JUDGE _____

INSTRUCTIONS FOR ATTORNEYS COMPLETING CIVIL COVER SHEET FORM JS 44

Authority For Civil Cover Sheet

The JS 44 civil cover sheet and the information contained herein neither replaces nor supplements the filings and service of pleading or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. Consequently, a civil cover sheet is submitted to the Clerk of Court for each civil complaint filed. The attorney filing a case should complete the form as follows:

- I.(a) Plaintiffs-Defendants.** Enter names (last, first, middle initial) of plaintiff and defendant. If the plaintiff or defendant is a government agency, use only the full name or standard abbreviations. If the plaintiff or defendant is an official within a government agency, identify first the agency and then the official, giving both name and title.
 - (b) County of Residence.** For each civil case filed, except U.S. plaintiff cases, enter the name of the county where the first listed plaintiff resides at the time of filing. In U.S. plaintiff cases, enter the name of the county in which the first listed defendant resides at the time of filing. (NOTE: In land condemnation cases, the county of residence of the "defendant" is the location of the tract of land involved.)
 - (c) Attorneys.** Enter the firm name, address, telephone number, and attorney of record. If there are several attorneys, list them on an attachment, noting in this section "(see attachment)".
- II. Jurisdiction.** The basis of jurisdiction is set forth under Rule 8(a), F.R.Cv.P., which requires that jurisdictions be shown in pleadings. Place an "X" in one of the boxes. If there is more than one basis of jurisdiction, precedence is given in the order shown below.
- United States plaintiff. (1) Jurisdiction based on 28 U.S.C. 1345 and 1348. Suits by agencies and officers of the United States are included here. United States defendant. (2) When the plaintiff is suing the United States, its officers or agencies, place an "X" in this box.
- Federal question. (3) This refers to suits under 28 U.S.C. 1331, where jurisdiction arises under the Constitution of the United States, an amendment to the Constitution, an act of Congress or a treaty of the United States. In cases where the U.S. is a party, the U.S. plaintiff or defendant code takes precedence, and box 1 or 2 should be marked.
- Diversity of citizenship. (4) This refers to suits under 28 U.S.C. 1332, where parties are citizens of different states. When Box 4 is checked, the citizenship of the different parties must be checked. (See Section III below; **NOTE: federal question actions take precedence over diversity cases.**)
- III. Residence (citizenship) of Principal Parties.** This section of the JS 44 is to be completed if diversity of citizenship was indicated above. Mark this section for each principal party.
- IV. Nature of Suit.** Place an "X" in the appropriate box. If there are multiple nature of suit codes associated with the case, pick the nature of suit code that is most applicable. Click here for: [Nature of Suit Code Descriptions](#).
- V. Origin.** Place an "X" in one of the seven boxes.
- Original Proceedings. (1) Cases which originate in the United States district courts.
- Removed from State Court. (2) Proceedings initiated in state courts may be removed to the district courts under Title 28 U.S.C., Section 1441.
- Remanded from Appellate Court. (3) Check this box for cases remanded to the district court for further action. Use the date of remand as the filing date.
- Reinstated or Reopened. (4) Check this box for cases reinstated or reopened in the district court. Use the reopening date as the filing date.
- Transferred from Another District. (5) For cases transferred under Title 28 U.S.C. Section 1404(a). Do not use this for within district transfers or multidistrict litigation transfers.
- Multidistrict Litigation – Transfer. (6) Check this box when a multidistrict case is transferred into the district under authority of Title 28 U.S.C. Section 1407.
- Multidistrict Litigation – Direct File. (8) Check this box when a multidistrict case is filed in the same district as the Master MDL docket.
- PLEASE NOTE THAT THERE IS NOT AN ORIGIN CODE 7.** Origin Code 7 was used for historical records and is no longer relevant due to changes in statute.
- VI. Cause of Action.** Report the civil statute directly related to the cause of action and give a brief description of the cause. **Do not cite jurisdictional statutes unless diversity.** Example: U.S. Civil Statute: 47 USC 553 Brief Description: Unauthorized reception of cable service.
- VII. Requested in Complaint.** Class Action. Place an "X" in this box if you are filing a class action under Rule 23, F.R.Cv.P.
- Demand. In this space enter the actual dollar amount being demanded or indicate other demand, such as a preliminary injunction.
- Jury Demand. Check the appropriate box to indicate whether or not a jury is being demanded.
- VIII. Related Cases.** This section of the JS 44 is used to reference related pending cases, if any. If there are related pending cases, insert the docket numbers and the corresponding judge names for such cases.

Date and Attorney Signature. Date and sign the civil cover sheet.

EXHIBIT N

**UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION**

VIRTAMOVE, CORP.,

Plaintiff,

v.

ORACLE CORPORATION,
Defendant.

Case No. 7:24-cv-00339

JURY TRIAL DEMANDED

**COMPLAINT FOR PATENT INFRINGEMENT AGAINST
ORACLE CORPORATION**

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, “Plaintiff” or “VirtaMove”) makes the following allegations against Defendant Oracle Corporation (“Defendant” or “Oracle”):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant’s unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058 (collectively, the “Asserted Patents”). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as Appzero Software Corp. (“Appzero”), which was established in 2010.

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repackage, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove's mission is to move and modernize the world's server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach \$2.1 billion in 2019 and \$4.3 billion in 2022—a compound annual growth rate (“CAGR”) of 30%. *See, e.g.*, <https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022>. Another source reported the application containers market had a market size of \$5.45 billion in 2024 and estimated it to reach \$19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.*, <https://www.mordorintelligence.com/industry-reports/application-container-market>.

6. On information and belief, Oracle is a Delaware corporation with a principal place of business at 2300 Oracle Way, Austin, Texas 78741. Oracle may be served with process through its registered agent, the Corporation Trust Company, Corporation Trust Center, 1209 Orange St., Wilmington, Delaware 19801.

JURISDICTION AND VENUE

7. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

9. Venue is proper in this District because Defendant resides in this District, has a regular and established place of business in this District, and has committed acts of infringement within this District.

COUNT I

INFRINGEMENT OF U.S. PATENT NO. 7,519,814

10. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

11. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 ('814 patent), titled "System for Containerization of Application Sets," issued on April 14, 2009. A true and correct copy of the '814 Patent is attached as **Exhibit 1**.

12. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., Oracle Cloud Infrastructure ("OCI")

and Oracle Kubernetes Engine (“OCE”), that directly infringe, literally and/or under the doctrine of equivalents, claims of the ’814 patent. The infringement of the Asserted Patents is also attributable to Defendant. Defendant directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

13. Defendant also knowingly and intentionally induces infringement of claims of the ’814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the ’814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and served on Defendant. Despite this knowledge of the ’814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the ’814 patent. Defendant does so, knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the ’814 patent, thereby specifically intending for and inducing its customers to infringe the ’814 patent through the customers’ normal and customary use of the Accused Products.

14. Defendant has also infringed, and continue to infringe, claims of the ’814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the ’814 patent, in

violation of 35 U.S.C. § 271(c).

15. The Accused Products satisfy all claim limitations of one or more claims of the '814 patent. A claim chart comparing independent claim 1 of the '814 patent to a representative Accused Product is attached as **Exhibit 2**, which is hereby incorporated by reference in its entirety.

16. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the '814 patent pursuant to 35 U.S.C. § 271.

17. As a result of Defendant's infringement of the '814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. VirtaMove has complied with the requirements of 35 U.S.C. § 287 and is not aware of any unmarked products that practice the claims of the '814 patent. In the alternative, either VirtaMove's product was marked before the filing of this lawsuit, or no requirement for marking applies.

COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

18. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

19. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 ('058 patent), titled "Computing System Having User Mode Critical System Elements as Shared Libraries," issued on August 24, 2010. A true and correct copy of the '058 patent is attached as **Exhibit 3**.

20. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products (“Accused Products”), such as, e.g., Oracle Cloud Infrastructure (“OCI”) and Oracle Kubernetes Engine (“OCE”), that directly infringe, literally and/or under the doctrine of equivalents. The infringement of the Asserted Patents is also attributable to Defendant. Defendant directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

21. Defendant also knowingly and intentionally induces infringement of claims of the ’058 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the ’058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and served. Despite this knowledge of the ’058 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the ’058 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the ’058 patent, thereby specifically intending for and inducing its customers to infringe the ’058 patent through the customers’ normal and customary use of the Accused Products.

22. Defendant has also infringed, and continue to infringe, claims of the ’058 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a

staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '058 patent, in violation of 35 U.S.C. § 271(c).

23. The Accused Products satisfy all claim limitations of one or more claims of the '058 patent. A claim chart comparing claim 1 of the '058 patent to a representative Accused Product is attached as **Exhibit 4**, which is hereby incorporated by reference in its entirety.

24. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the '058 patent pursuant to 35 U.S.C. § 271.

25. As a result of Defendant's infringement of the '058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant's infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. VirtaMove has complied with the requirements of 35 U.S.C. § 287. 35 U.S.C. § 287 requires the marking of a "patented articles," however VirtaMove makes, offers for sale, and/or sells software and software support services. VirtaMove is unaware of instances where it made, offered for sale, and/or sold "a processor," as required by claim 1 of the '058 patent, with these products. Thus, VirtaMove is not required to mark any of these products that it makes, offers for sale, and/or sells. In the alternative, to the extent VirtaMove is found to have "made," "offered for sale," and/or "sold" "patented articles," including to the extent VirtaMove is found to have made, offered for sale, and/or sold products with "a processor" as required by claim 1 of the '058 patent, VirtaMove's products were marked before the filing of this lawsuit.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

- a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;
- b. A permanent injunction prohibiting Defendant from further acts of infringement of each of the '814 and '058 patents;
- c. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant's infringement of each of the Asserted Patents;
- d. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;
- e. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys' fees against Defendant; and
- f. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury of any issues so triable by right.

Dated: December 20, 2024

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)

rmirzaie@raklaw.com

Marc A. Fenster (CA SBN 181067)

mfenster@raklaw.com

Neil A. Rubin (CA SBN 250761)

nrubin@raklaw.com

Amy E. Hayden (CA SBN 287026)

ahayden@raklaw.com

Jacob R. Buczko (CA SBN 269408)

jbuczko@raklaw.com

James S. Tsuei (CA SBN 285530)

jtsuei@raklaw.com

James A. Milkey (CA SBN 281283)

jmilkey@raklaw.com

Christian W. Conkle (CA SBN 306374)

cconkle@raklaw.com

Jonathan Ma (CA SBN 312773)

jma@raklaw.com

Daniel Kolko (CA SBN 341680)

dkolko@raklaw.com

Mackenzie Paladino (NY SBN 6039366)

mpaladino@raklaw.com

RUSS AUGUST & KABAT

12424 Wilshire Boulevard, 12th Floor

Los Angeles, CA 90025

Telephone: (310) 826-7474

Qi (Peter) Tong (TX SBN 24119042)

ptong@raklaw.com

RUSS AUGUST & KABAT

8080 N. Central Expy., Suite 1503

Dallas, TX 75206

Telephone: (310) 826-7474

Attorneys for Plaintiff VirtaMove, Corp.

Exhibit 1



US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,519,814 B2**
(45) **Date of Patent:** **Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

WO WO 2006/039181 A 4/2006

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**
US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data
(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)
(52) **U.S. Cl.** **713/167**; 713/164; 709/248;
709/214; 719/319
(58) **Field of Classification Search** 713/167;
719/319
See application file for complete search history.

(56) **References Cited**
U.S. PATENT DOCUMENTS
6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

OTHER PUBLICATIONS

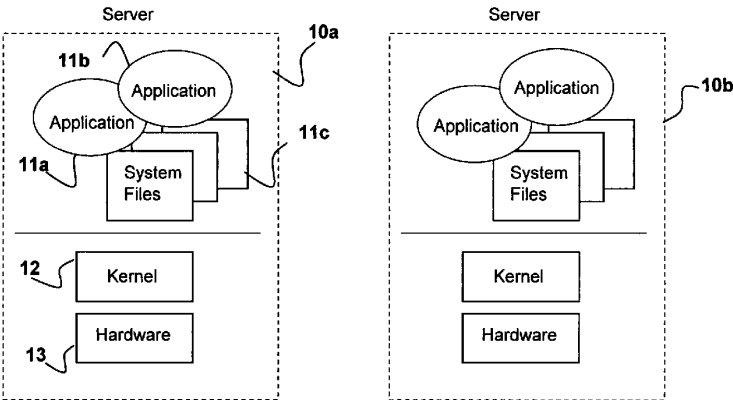
Soltész, S., et al, 'Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=4667.*

Primary Examiner—Kambiz Zand
Assistant Examiner—Ronald Baum
(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



US 7,519,814 B2

Page 2

U.S. PATENT DOCUMENTS				2002/0004854 A1	1/2002	Hartley	
6,847,970 B2 *	1/2005	Keller et al.	707/100	2002/0174215 A1	11/2002	Schaefer	709/224
7,076,784 B1 *	7/2006	Russell et al.	719/315	2003/0101292 A1	5/2003	Fisher	
7,287,259 B2 *	10/2007	Grier et al.	719/331	* cited by examiner			

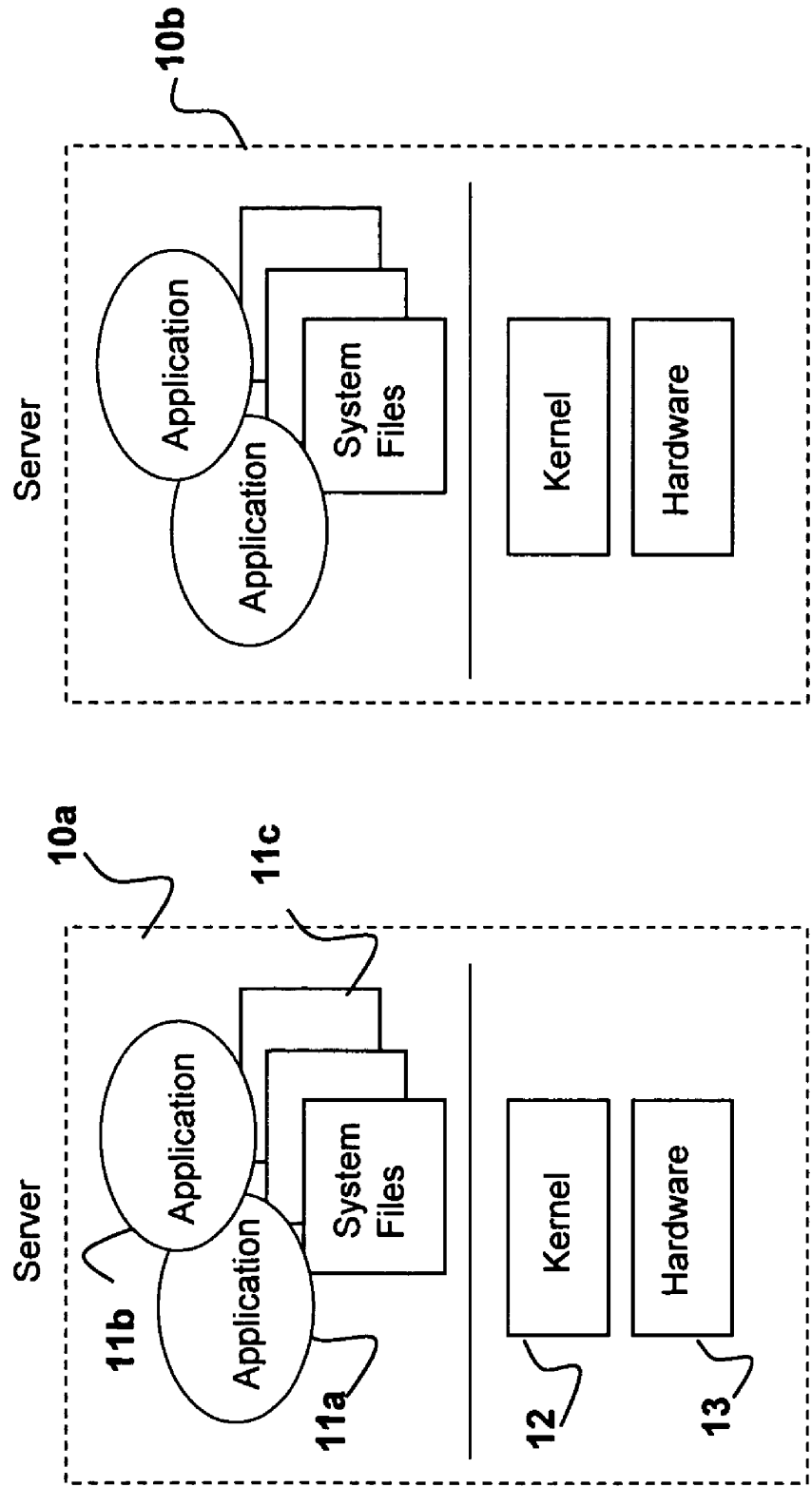


Figure 1

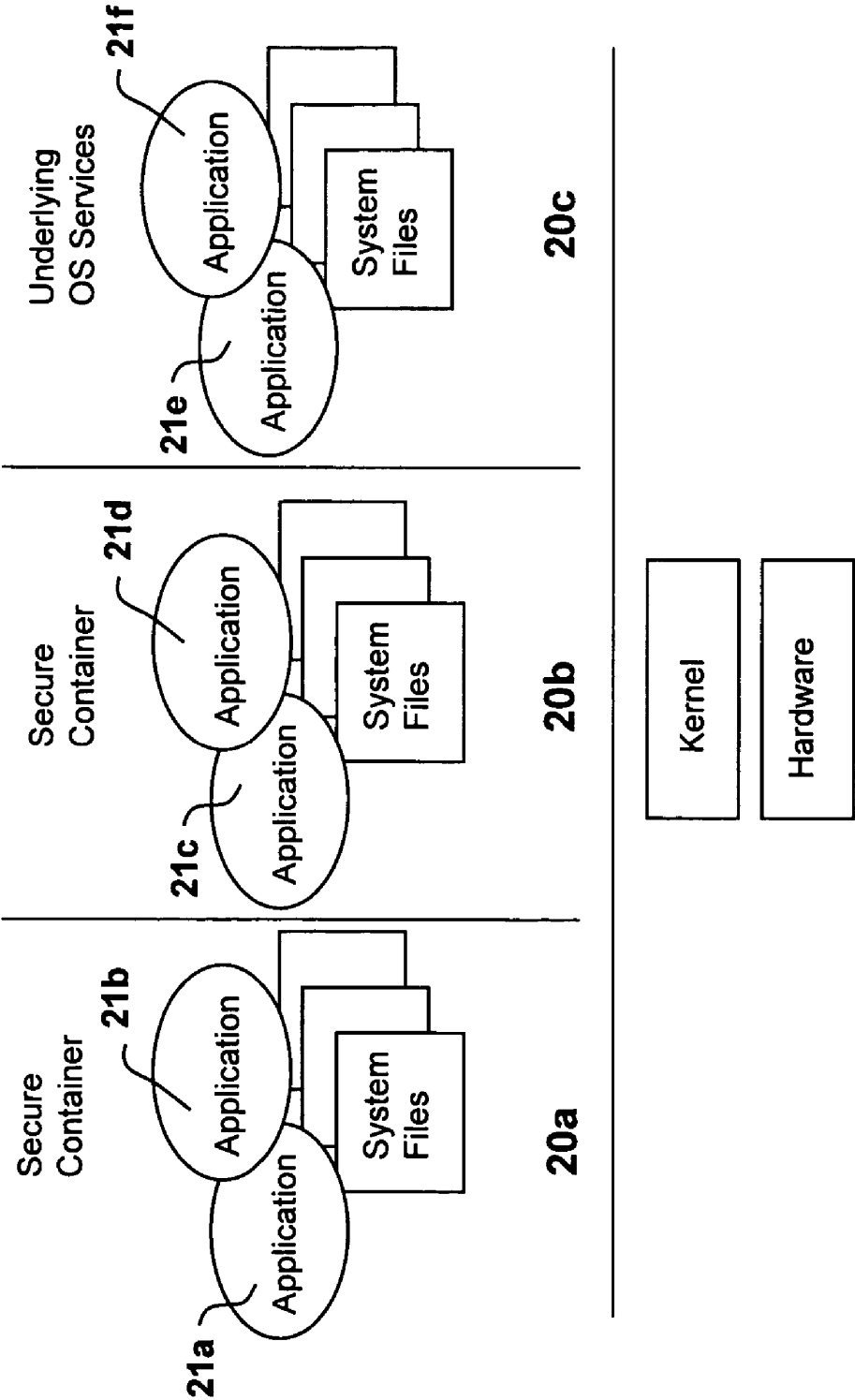


Figure 2

U.S. Patent

Apr. 14, 2009

Sheet 3 of 17

US 7,519,814 B2

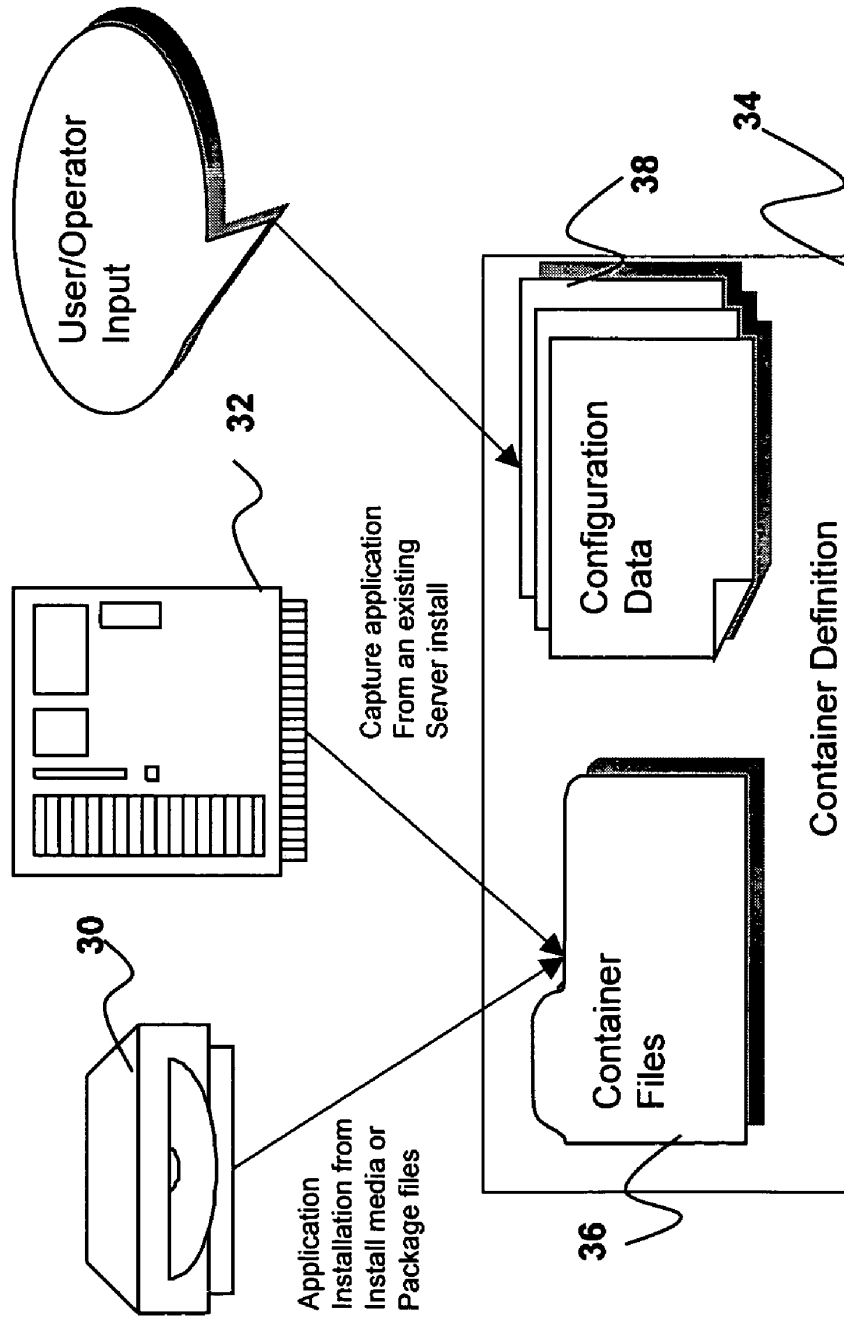


Figure 3

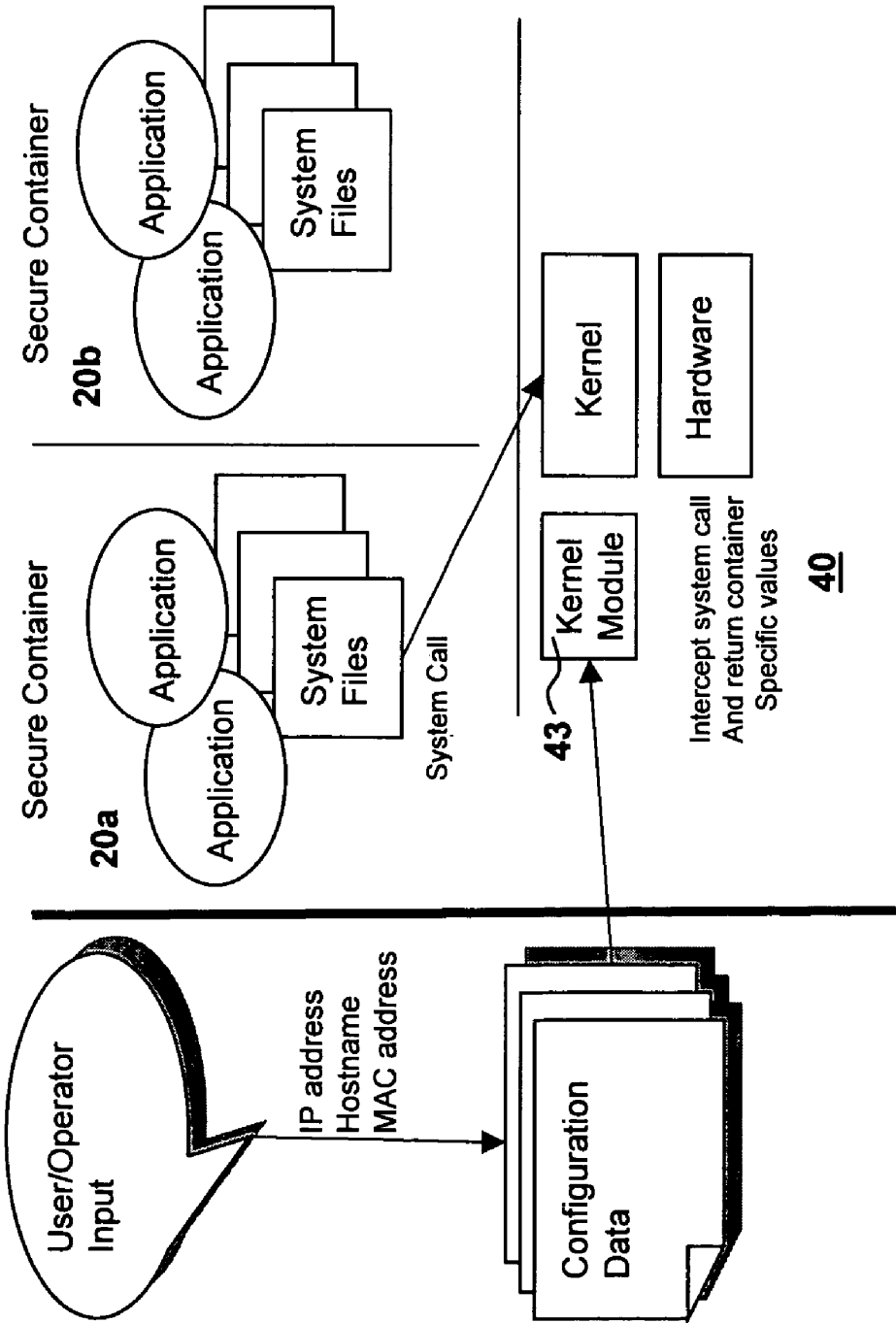


Figure 4

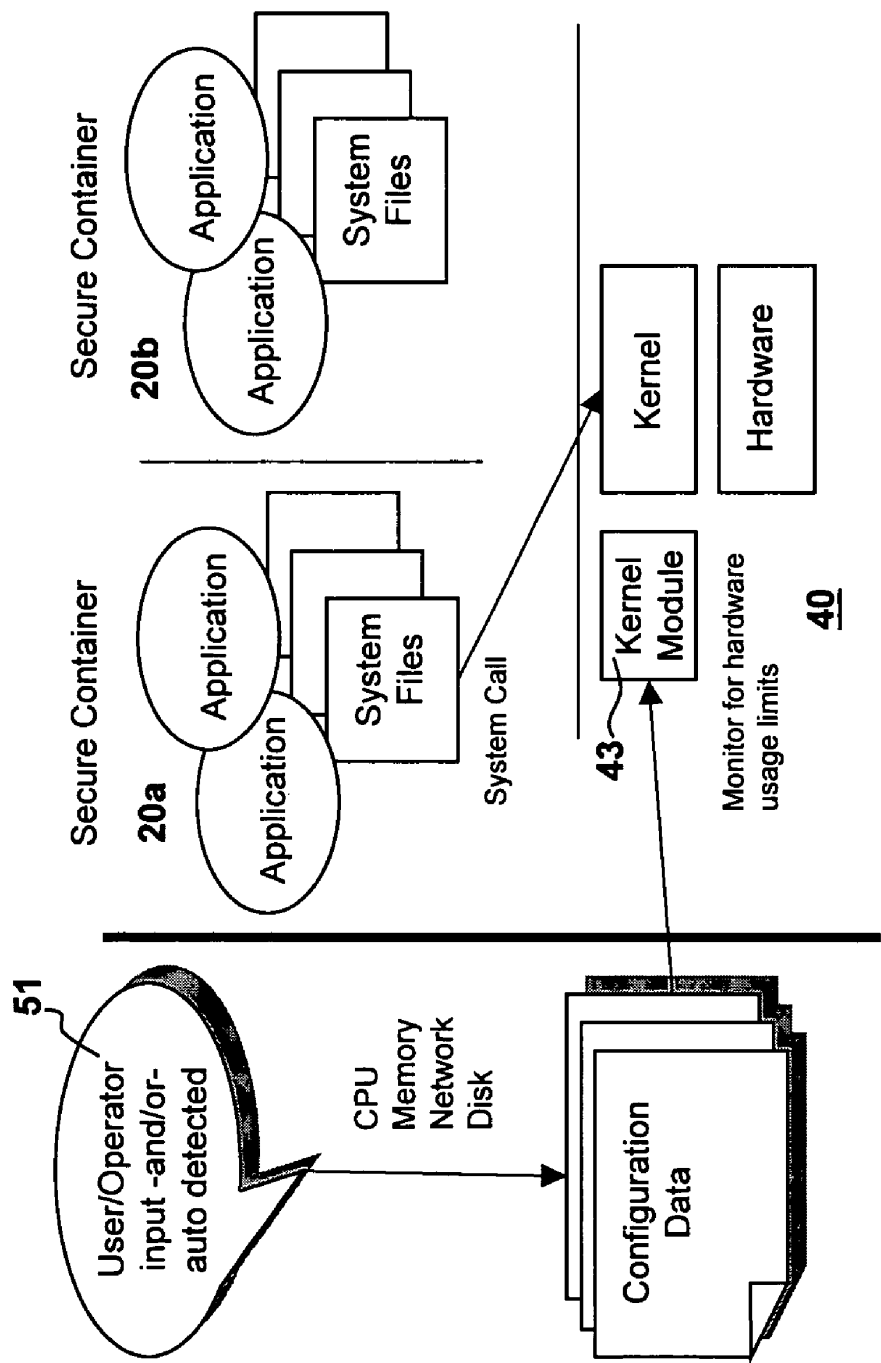
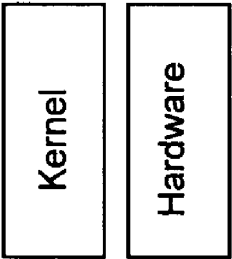
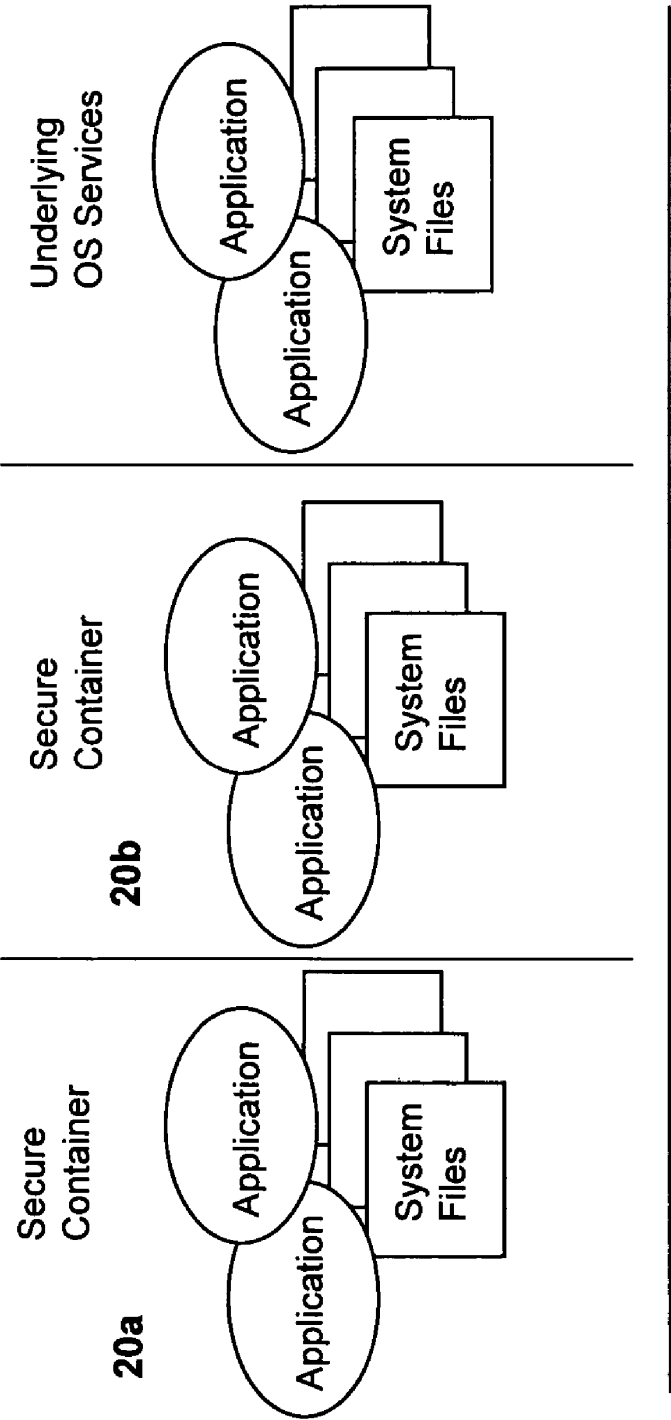


Figure 5



40

Figure 6

U.S. Patent

Apr. 14, 2009

Sheet 7 of 17

US 7,519,814 B2

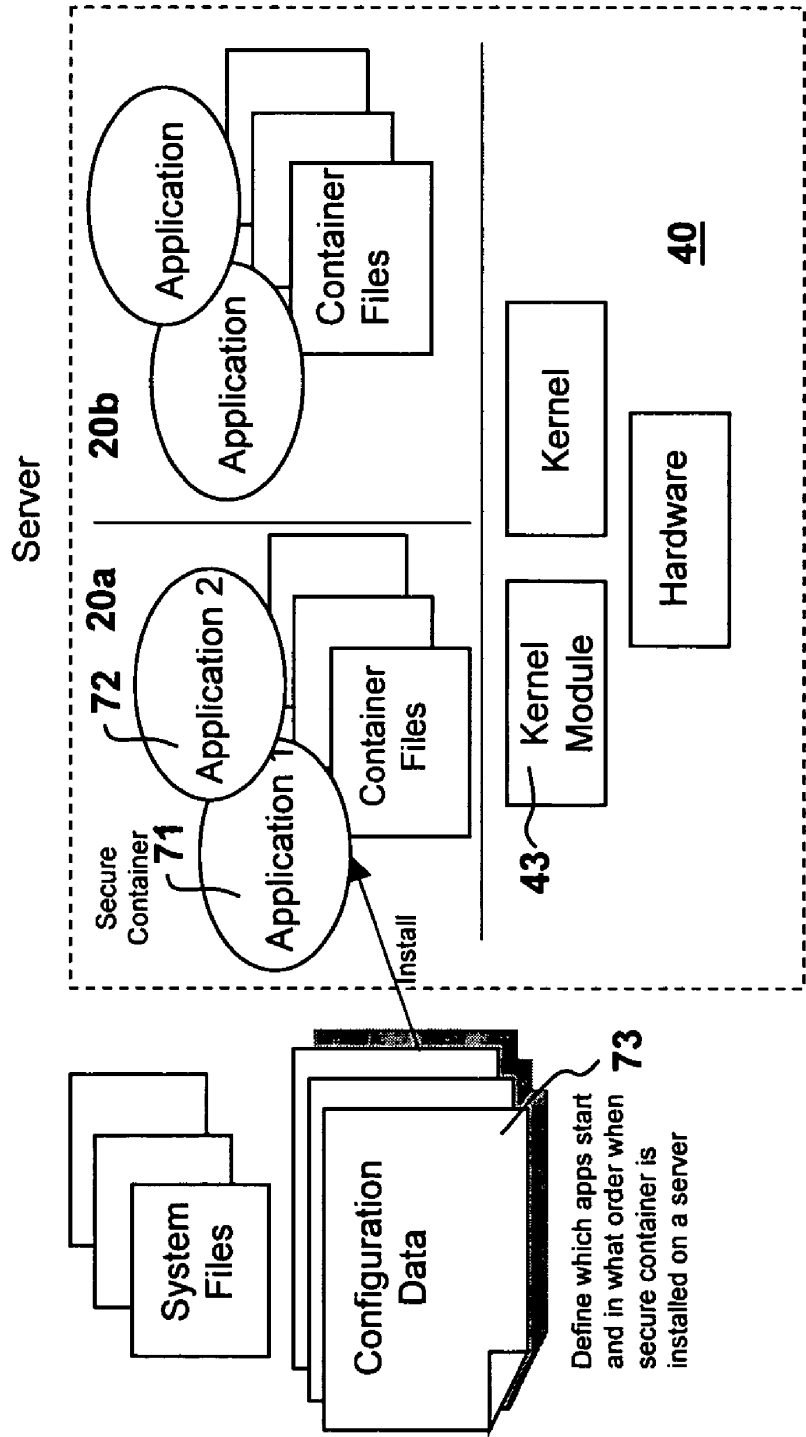


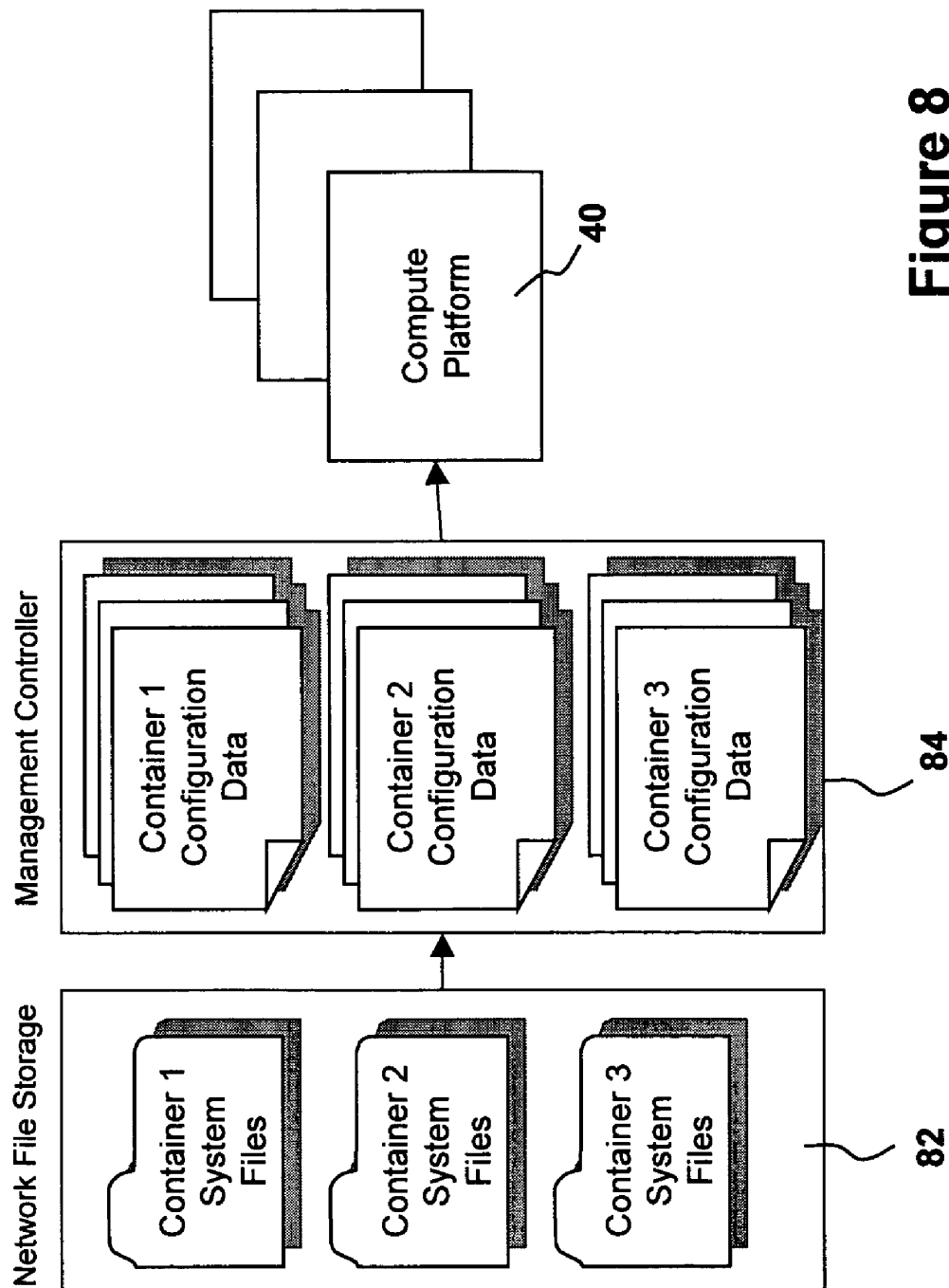
Figure 7

U.S. Patent

Apr. 14, 2009

Sheet 8 of 17

US 7,519,814 B2



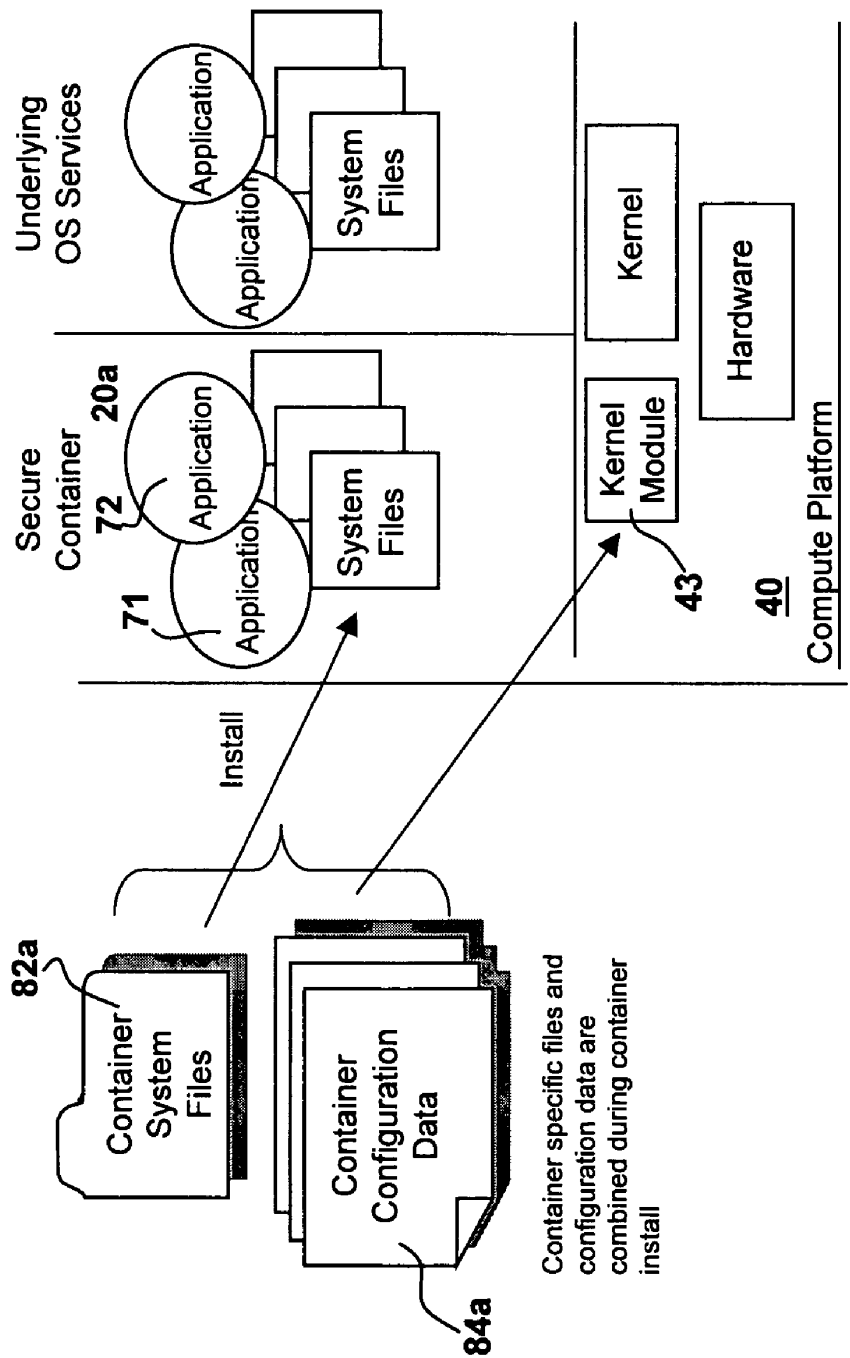


Figure 9

U.S. Patent

Apr. 14, 2009

Sheet 10 of 17

US 7,519,814 B2

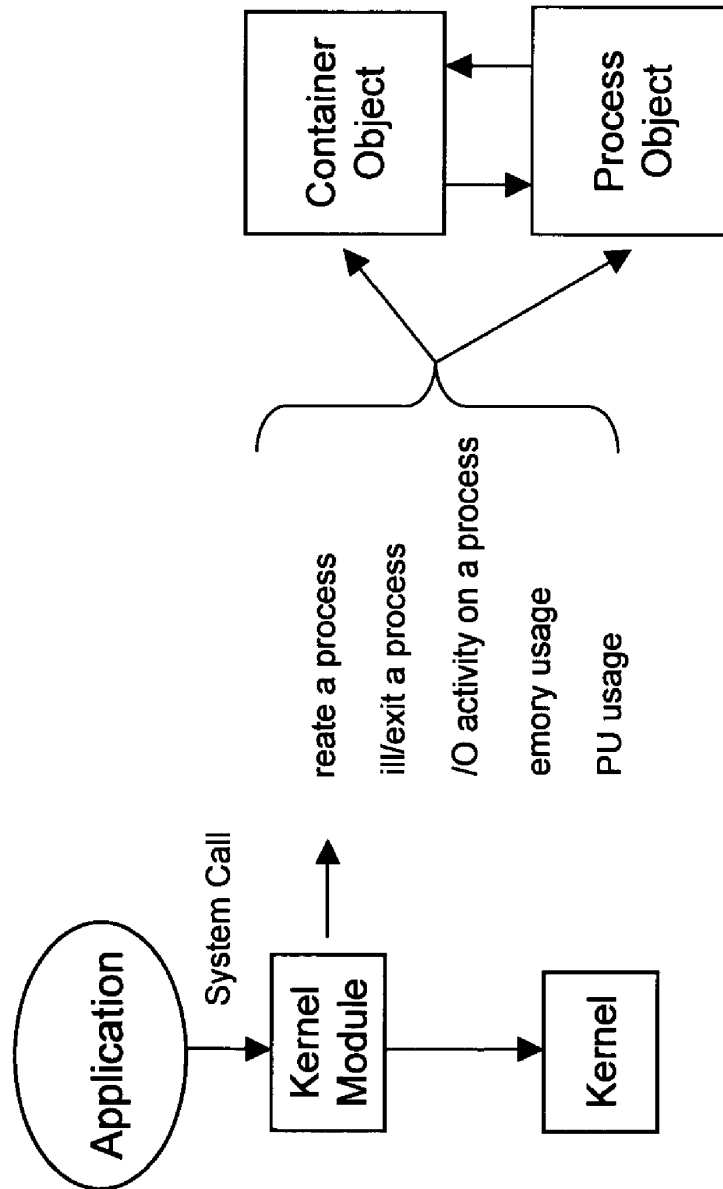


Figure 10

U.S. Patent

Apr. 14, 2009

Sheet 11 of 17

US 7,519,814 B2

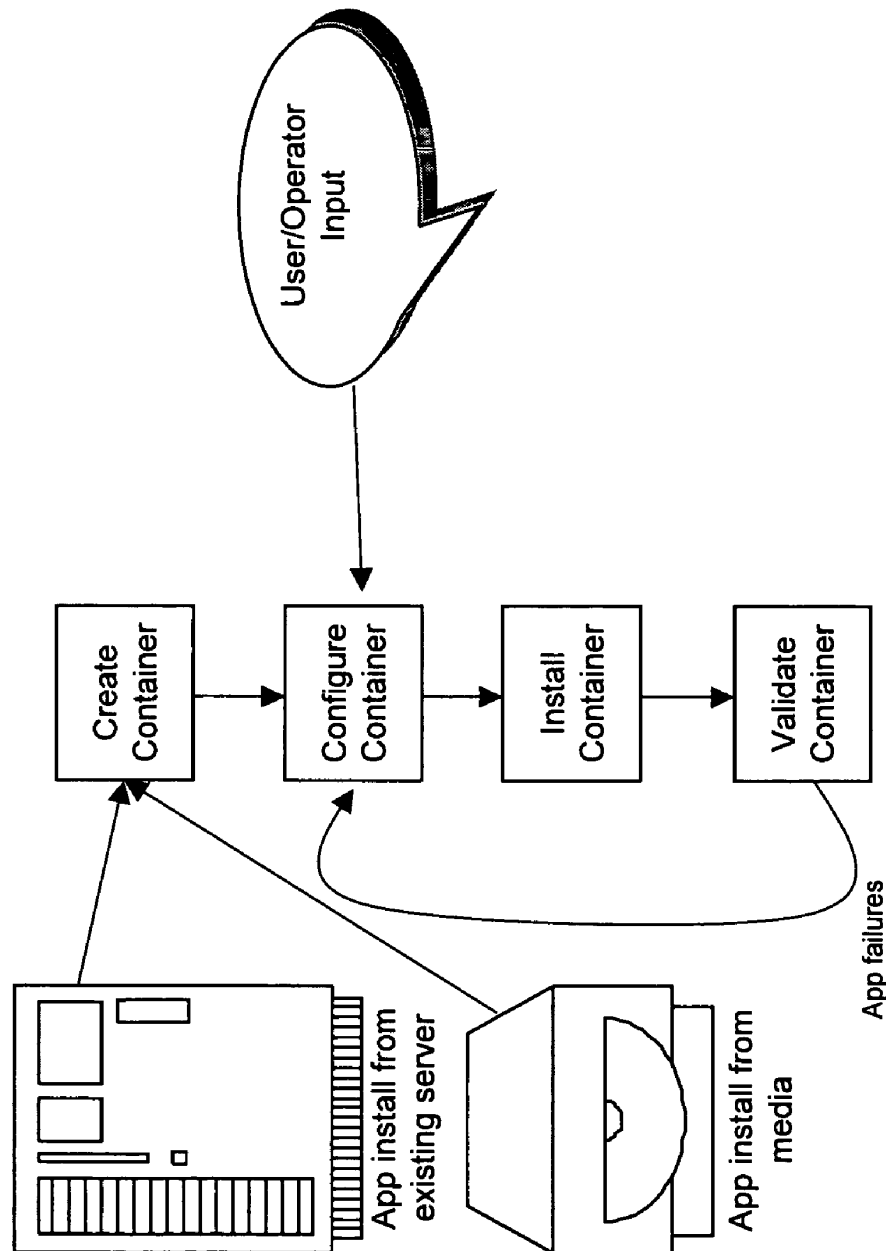


Figure 11

U.S. Patent

Apr. 14, 2009

Sheet 12 of 17

US 7,519,814 B2

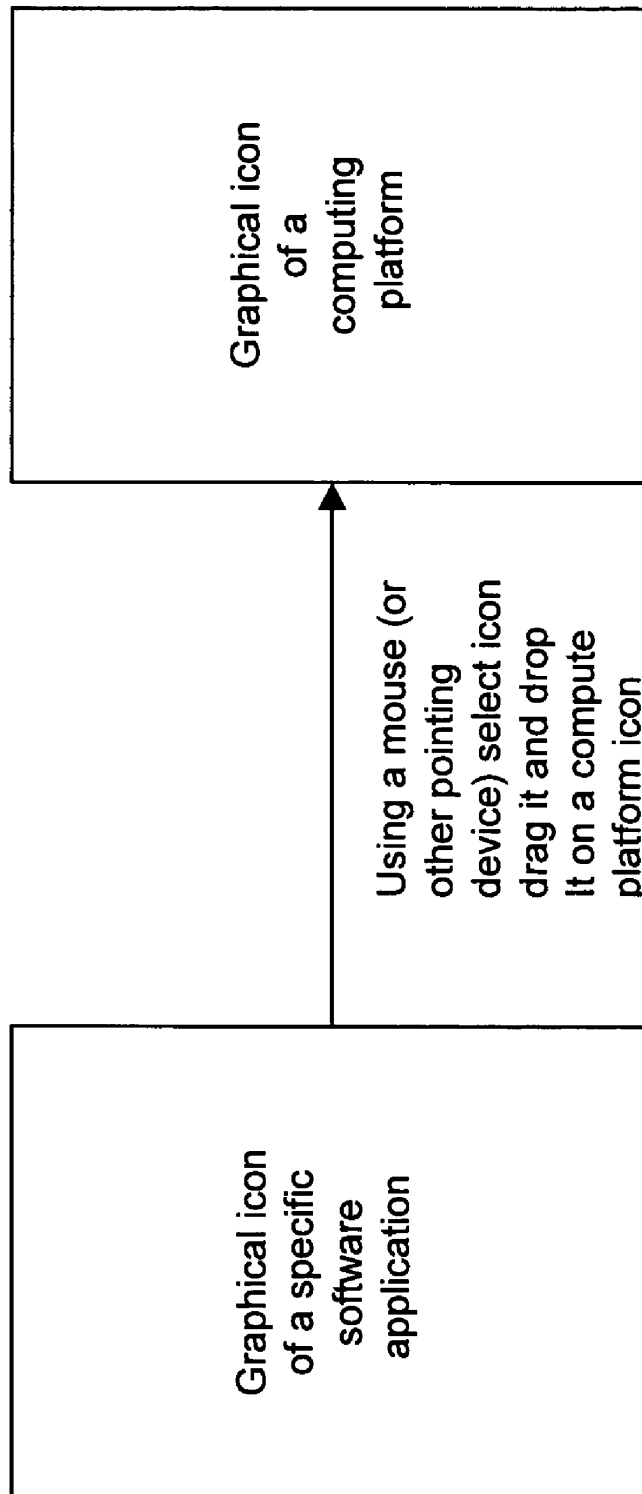


Figure 12

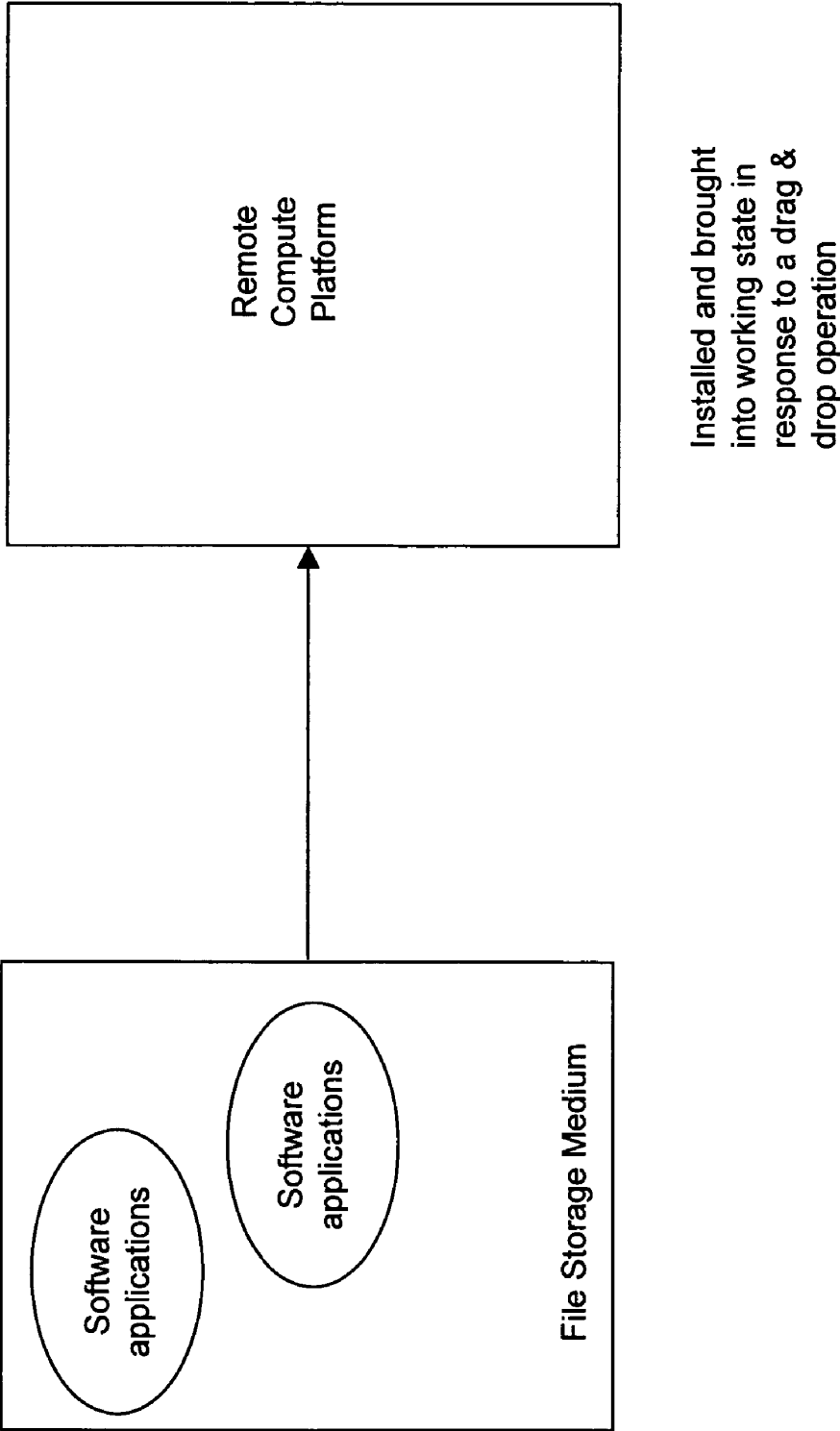


Figure 13

U.S. Patent

Apr. 14, 2009

Sheet 14 of 17

US 7,519,814 B2

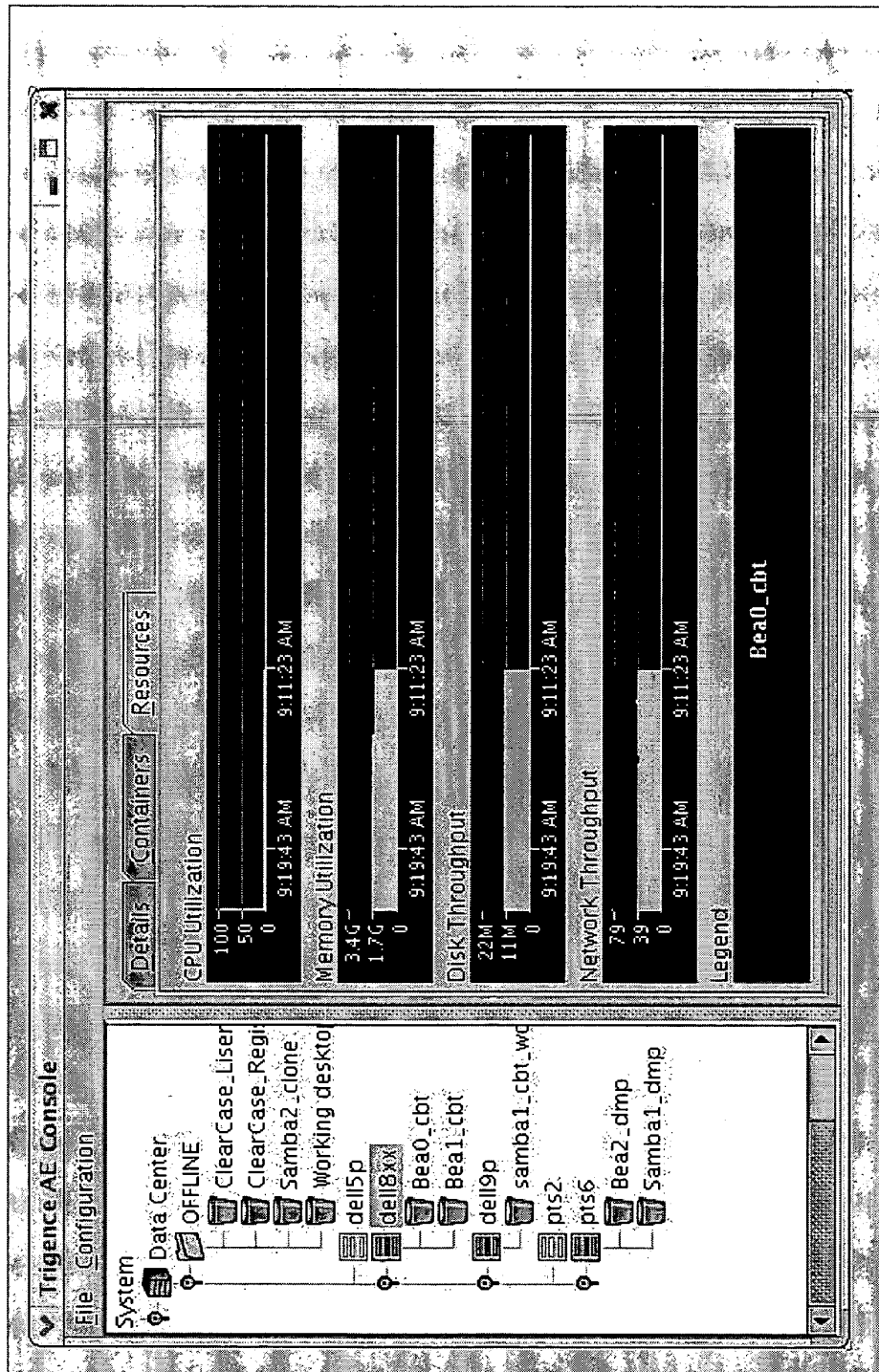


Figure 14

U.S. Patent

Apr. 14, 2009

Sheet 15 of 17

US 7,519,814 B2

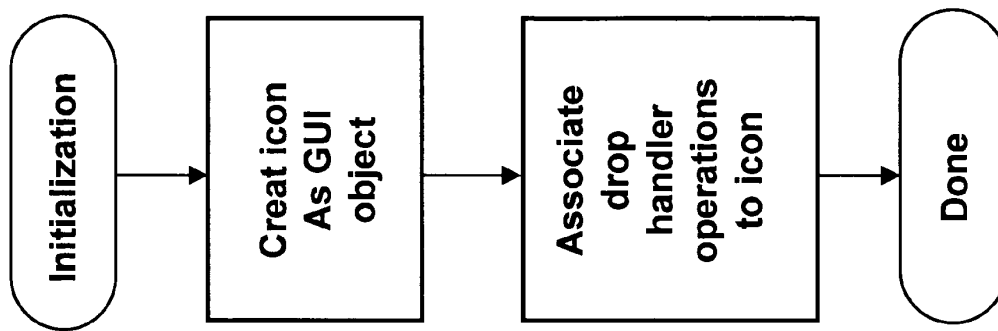


Figure 15

U.S. Patent

Apr. 14, 2009

Sheet 16 of 17

US 7,519,814 B2

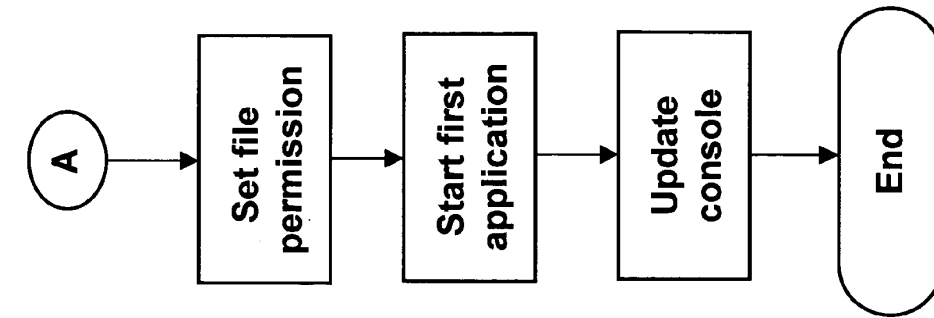
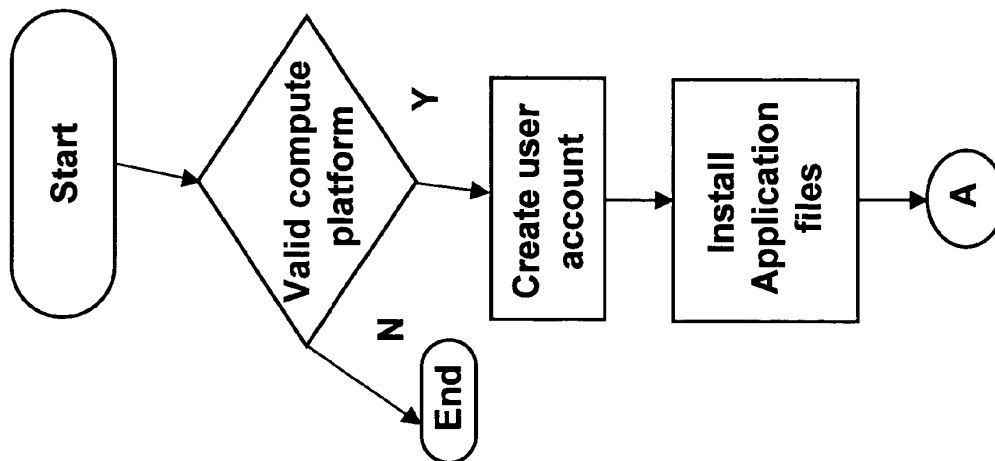


Figure 16



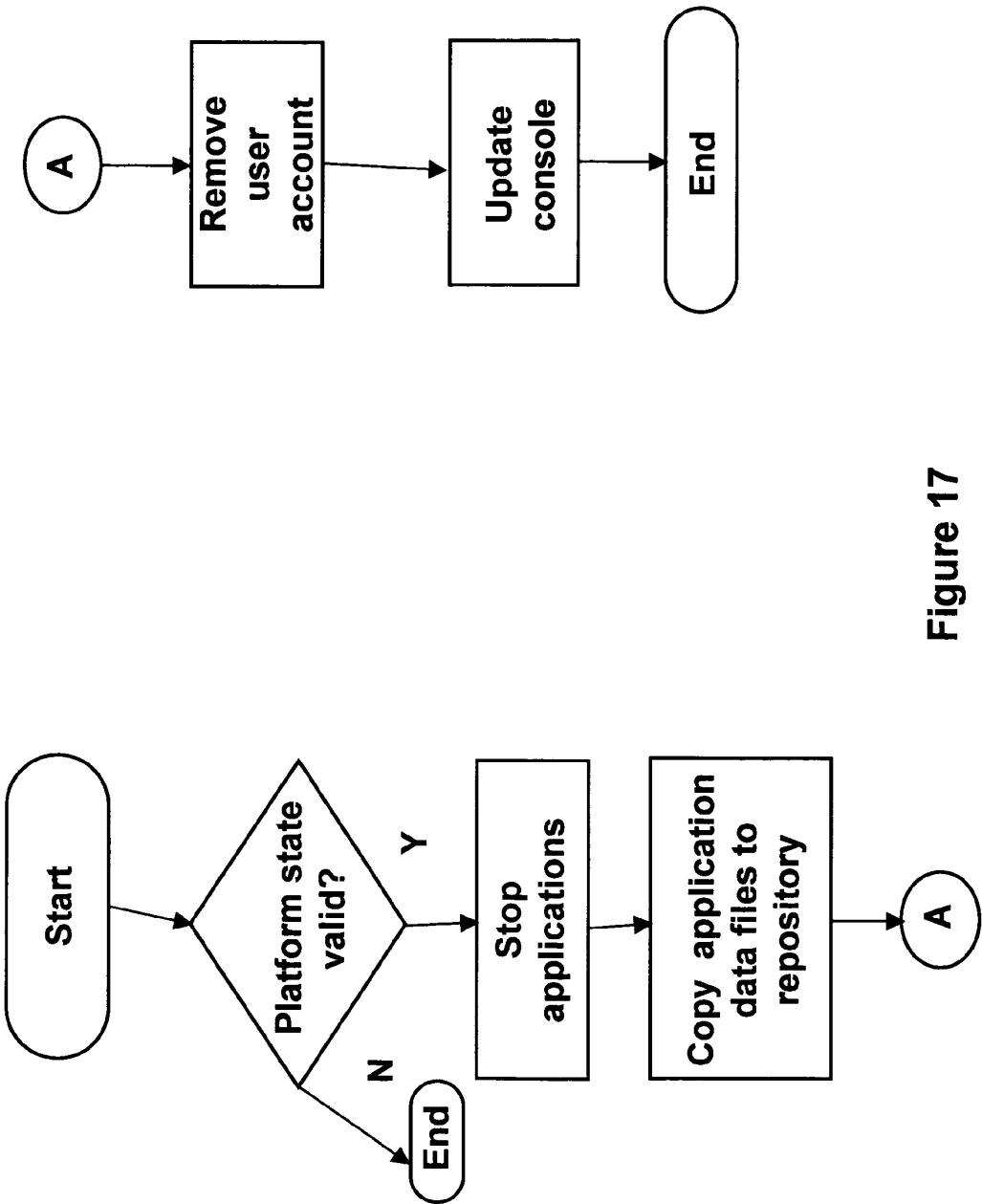


Figure 17

US 7,519,814 B2

1

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

2

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

```
/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1
```


US 7,519,814 B2

3

/lib/libnsl.so.1
 /lib/libdl.so.2
 /lib/i686/libpthread.so.0
 /lib/i686/libc.so.6
 /lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
 /etc/httpd/conf
 /etc/httpd/conf.d
 /etc/httpd/logs
 /etc/httpd/modules
 /etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications

4

associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySQL and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

US 7,519,814 B2

5

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of

6

this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed “spoofing” and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. 1 is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. 2 is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. 3 is a pictorial diagram illustrating the creation of a container.

FIG. 4 is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the “kernel module” which is used to handle system calls.

FIG. 5 is a diagram similar to FIG. 4, wherein additional configuration data is provided.

FIG. 6 is a diagram illustrating a system wherein the containers are secure containers.

FIG. 7 is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. 8 is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. 9 is a diagram that illustrates the installation of a container on a server.

FIG. 10 is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. 11 is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. 12 is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. 13 is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. 14 is a screen snapshot of an implementation according to the schematic of FIG. 13.

FIG. 15 is a flow chart of a method of initializing icons of FIG. 14.

FIG. 16 is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. 14; and,

FIG. 17 is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13.

DETAILED DESCRIPTION

Turning now to FIG. 1, a system is shown having a plurality of servers 10a, 10b. Each server includes a processor and an independent operating system. Each operating system includes a kernel 12, hardware 13 in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs 11a, 11b and related system files 11c.

US 7,519,814 B2

7

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. 2 illustrates a system in accordance with the invention in which multiple applications **21a**, **21b**, **21c**, **21d**, **21e**, and **21f** can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20a**, **20b** and **20c**. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20a**, **20b** and **20c** are shown in FIG. 2 in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. 3 the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. 4 illustrates a system having two secure containers **20a** **20b**, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20a**, **20b** environment system calls are intercepted, by the kernel module **43** passing through services installed as part of

8

this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by “spoofing” the system.

As introduced heretofore, “spoofing” is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system’s kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the ‘uname’ system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has ‘spoofed’ the ‘uname’ system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same “spoofing” mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

- tracking applications, for example a tracking of which applications are in and out of a specific container;
- spoofing particular values returned by the kernel;
- applying resource checks and imposing resource limits;
- monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,
- retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. 5 including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20a** or **20b**. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. 6 that each application executing associated with a container **20a** or **20b**, or contained within a

US 7,519,814 B2

9

container **20a** or **20b**, is able to access files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20a** association are not able to access the files contained in another container **20b**.

When a container **20a** or **20b** is installed specific applications are started, as is shown in FIG. 7 and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. 7, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. 8, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. 9 shows that the files **82a**, **84a** are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing

10

the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. 9.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. 10 shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. 3 illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform and then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. 3, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. 3. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

US 7,519,814 B2

11

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

- modifying the contents a file to add a container specific IP address;
- modifying the contents of a directory to define start scripts that should be executed;
- modifying the contents of a file to define which mount points will relate to a container;
- removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications

12

are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

US 7,519,814 B2

13

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

14

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations

Icons

Storage medium

Console

Network connections

One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. 12, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. 13, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

US 7,519,814 B2

15

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. 14, a screen snapshot of an implementation is shown according to the schematic of FIG. 13. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. 14, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. 15, shown is a flow chart of a method of initializing the icons of FIG. 14. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. 12 to 14, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. 16 which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. 2. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. 16 are performed by an install drop handler. During installation, verification is performed to

16

determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. 17, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. 15), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. 17 which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. 13. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

US 7,519,814 B2

17

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. 16 and 17, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim 1, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim 2, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

18

4. A method as defined in claim 1 further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim 2, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim 2, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim 2 further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim 1, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim 2, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim 2, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim 2, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim 11, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim 1.

13. A method as defined in claim 1 further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

- a) running an instance of a service on a server;
- b) determining which files are being used; and,
- c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim 14 comprising the steps of:

- assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;
- installing the container on a server; and,
- testing the applications and files within the container.

16. A method as defined in claim 1 comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

- using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim 1 further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes: using a graphical user interface, associating a unique icon representing a service with a unique icon representing

US 7,519,814 B2

19

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim 17 wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim 18, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim 19, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim 20 further comprising starting a distributed software application.

22. A method according to anyone of claims 20 further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim 17, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim 17 further comprising creating a user account for the service.

25. A method as defined in claim 17, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim 17 further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim 1, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;
displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim 27 further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim 27 further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim 27 further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

20

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim 31, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim 32, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim 31, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and, forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *







Exhibit 2

U.S. Patent No. 7,519,814 vs. Oracle

Accused Instrumentalities: Oracle products and services using secure containerized applications, including without limitation Oracle Cloud Infrastructure (“OCI”) and Oracle Kubernetes Engine (“OKE”), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
<p>[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:</p>	<p>To the extent the preamble is limiting, Oracle and/or its customer practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed.</p> <p>For example, Oracle Kubernetes Engine runs on individual servers, each of which runs an independent operating system running either on bare metal, through an on-premises virtualized infrastructure, through one or more cloud services, or through any other supported deployment. In an exemplary deployment, two or more servers use different operating systems. The servers operate in disparate computing environments, including because each server is a stand-alone computer and/or each server is unrelated to the other servers due to having independent hardware and, in some instances, independent software.</p> <p>Oracle requires and/or provides that each server includes a processor with one or more cores available to the OS kernel. Oracle further requires and/or provides that each server has a supported operating system, which includes a kernel and associated local system files, including for example libraries such as libc/glibc, configuration files, etc. In the infringing system, at least two servers have different operating systems.</p> <p>In at least some instances, Oracle directly owns, operates, controls, and/or benefits from the claimed system and/or method. In other instances, Oracle’s customer makes and uses the system and/or method either by following Oracle’s direction and control, including Oracle’s documentation, or automatically through the ordinary and expected operation of Oracle’s software, or a combination thereof.</p>

Claim 1	Accused Instrumentalities
	<p><i>See claim limitations below.</i></p> <p><i>See also, e.g.:</i></p> <h2 data-bbox="737 378 1106 427">Why Choose OKE?</h2> <div data-bbox="732 492 1902 1141"> <div data-bbox="732 492 1106 792">  <p>Price-Performance</p> <p>OKE is the lowest cost Kubernetes service amongst all hyperscalers, especially for serverless.</p> </div> <div data-bbox="1142 492 1516 792">  <p>Autoscaling</p> <p>OKE automatically adjusts compute resources based on demand, which can reduce your costs.</p> </div> <div data-bbox="1551 492 1902 792">  <p>Efficiency</p> <p>GPUs can be scarce, but OKE job scheduling makes it easy to maximize resource utilization.</p> </div> <div data-bbox="732 849 1106 1141">  <p>Portability</p> <p>OKE is consistent across clouds and on-premises, enabling portability and avoiding vendor lock-in.</p> </div> <div data-bbox="1142 849 1516 1141">  <p>Simplicity</p> <p>OKE reduces the time and cost needed to manage the complexities of Kubernetes infrastructure.</p> </div> <div data-bbox="1551 849 1902 1141">  <p>Reliability</p> <p>Automatic upgrades and security patching boost reliability for the control plane and worker nodes.</p> </div> </div> <p data-bbox="711 1182 1467 1214">https://www.oracle.com/cloud/cloud-native/kubernetes-engine/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 272 1671 326">Welcome to Oracle Cloud Infrastructure</p> <p data-bbox="730 370 1902 511">Oracle Cloud Infrastructure (OCI) is a set of complementary cloud services that enable you to build and run a range of applications and services in a highly available hosted environment. OCI provides high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network.</p> <p data-bbox="714 532 1642 560">https://docs.oracle.com/en-us/iaas/Content/GSG/Concepts/baremetalintro.htm</p> <p data-bbox="756 618 1835 722">Existing applications can benefit by migrating to OCI and OKE</p> <p data-bbox="756 771 1764 803">OKE offers lower total cost of ownership and improved time to market.</p> <p data-bbox="756 863 1556 896">OKE simplifies operations at scale in the following ways:</p> <ul data-bbox="756 979 1835 1333" style="list-style-type: none"> <li data-bbox="756 979 1251 1057">– Lift and shift; there's no need to rearchitect <li data-bbox="756 1117 1243 1195">– Reduce operations burden with automation <li data-bbox="756 1255 1184 1333">– Save time on infrastructure management <li data-bbox="1318 979 1818 1057">– Increase resource utilization and efficiency <li data-bbox="1318 1117 1835 1195">– Improve agility, flexibility, uptime, and resilience <li data-bbox="1318 1255 1755 1333">– Reduce compliance risk and enhance security <p data-bbox="714 1377 1650 1404">https://www.oracle.com/cloud/cloud-native/kubernetes-engine/#app-migration</p>

Claim 1	Accused Instrumentalities
	<p>What is OCI Kubernetes Engine (OKE)?</p> <p>Oracle Cloud Infrastructure Kubernetes Engine (OKE) is a managed Kubernetes service that simplifies the development, deployment, and operation of containerized workloads at scale. OKE enables you to quickly create, manage, and consume Kubernetes clusters that leverage underlying OCI compute, networking, and storage services.</p> <p>When should I use OKE?</p> <p>You should use OKE when you want to leverage Kubernetes to deploy and manage your Kubernetes-based container applications. It allows you to combine the production-grade container orchestration of standard upstream Kubernetes with the control, security, and high, predictable performance of OCI.</p> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p> <p>How does OKE provide resiliency?</p> <p>When you create an OKE cluster, OKE automatically creates and manages multiple Kubernetes control plane nodes spread across fault domains and availability domains (logical data centers). This is done to help ensure that the managed Kubernetes control plane is highly available. Control plane operations, such as upgrading to newer versions of Kubernetes, can be performed without service interruptions. Additionally, when you provision worker nodes, you can use a placement configuration to control the fault domain and availability domain where they are created. Nodes will automatically come online with labels, which you can use to schedule your workloads so they are robust and highly available.</p> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p> <p>Can I deploy private Kubernetes clusters?</p> <p>Yes; with OKE, your Kubernetes clusters are integrated in your VCN. Your cluster worker nodes, load balancers, and the Kubernetes API endpoint are part of a private or public subnet of your VCN. Regular VCN routing and firewall rules control access to the Kubernetes API endpoint, making it accessible from a corporate network only, through a bastion host, or by specific platform services.</p> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p>

Claim 1	Accused Instrumentalities
	<p>When should I use virtual nodes, managed nodes, or self-managed nodes?</p> <ul style="list-style-type: none"> Virtual nodes Virtual nodes offer a serverless Kubernetes experience. This option is ideal if you'd rather focus on your application and avoid managing the underlying infrastructure. Virtual nodes relieve you of management-related tasks such as scaling, upgrading, patching, troubleshooting, and provisioning worker nodes. Managed nodes Managed nodes are a good choice for general purpose workloads. They offer an extensive list of customizable configuration options that have been tested by the OKE service. Unlike fully managed virtual nodes, you share the management of worker nodes with OCI. OKE simplifies the management process through features such as on-demand cycling to automate worker node updates, cluster self-healing upon failure detection, autoscaling, and more. Self-managed nodes Self-managed nodes offer access to the underlying infrastructure, configuration options, and compute shapes that aren't currently available to managed nodes. This includes access to specialized infrastructure, such as RDMA-enabled bare metal cluster networks or confidential compute shapes. This advanced control makes self-managed nodes ideal for specialized use cases that aren't supported with managed nodes. Note that with self-managed nodes, you are fully responsible for managing the worker nodes—without the automated features provided by managed or virtual nodes. <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p> <p>What are the storage options for virtual nodes?</p> <p>OKE virtual nodes do not yet have persistent storage capabilities. However, there are plans to introduce support for attaching persistent volumes backed by OCI Block Storage and OCI File Storage. If your Kubernetes application requires persistent storage, it's advisable to use OKE managed nodes.</p> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="747 272 1562 321">Supported Images for Managed Nodes</h2> <p data-bbox="747 363 1919 427">Kubernetes Engine supports the provisioning of worker nodes (managed nodes only) using some, but not all, of the latest Oracle Linux images provided by Oracle Cloud Infrastructure.</p> <p data-bbox="747 469 1583 493">You can use these Oracle Linux images when provisioning managed nodes:</p> <ul data-bbox="772 535 991 667" style="list-style-type: none"><li data-bbox="772 535 945 560">• <a data-bbox="810 535 945 560" href="#">OKE Images<li data-bbox="772 586 991 610">• <a data-bbox="810 586 991 610" href="#">Platform Images<li data-bbox="772 636 982 660">• <a data-bbox="810 636 982 660" href="#">Custom Images <p data-bbox="714 703 1785 727"><a data-bbox="714 703 1785 727" href="https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengimagesshapes.htm">https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengimagesshapes.htm</p> <h2 data-bbox="722 769 1033 812">What is Docker?</h2> <p data-bbox="722 911 1902 1057">A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly.</p> <p data-bbox="722 1089 1919 1175">Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle.</p> <p data-bbox="722 1208 1860 1294">Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical <a data-bbox="835 1240 961 1265" href="#">open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products.</p> <p data-bbox="714 1320 1680 1344"><a data-bbox="714 1320 1680 1344" href="https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/">https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>Docker engine:</p> <p>The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu.</p> <p>Docker images:</p> <p>Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image.</p> <p>Docker Registry:</p> <p>Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p> <p>Docker versus Kubernetes</p> <p>Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

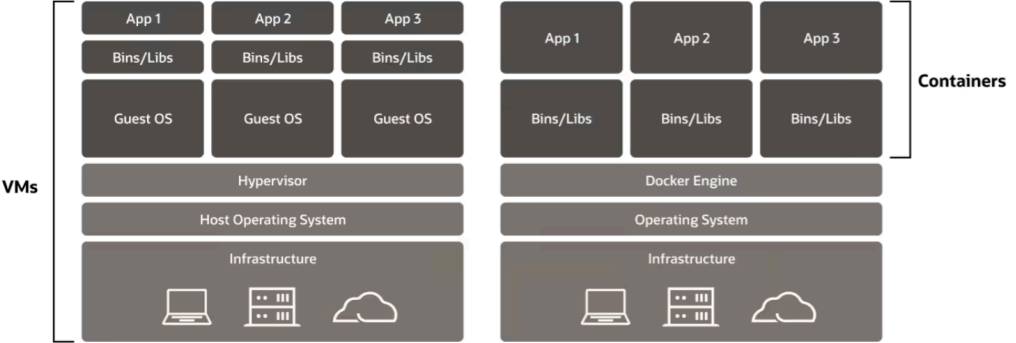
Claim 1	Accused Instrumentalities
	<p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p> <div style="display: flex; justify-content: space-around;"> <div data-bbox="783 558 1245 899"> <p>Virtual Machines</p> <ul style="list-style-type: none"> Each virtual machine (VM) includes the app, the necessary binaries and libraries and an <u>entire guest operating system</u> </div> <div data-bbox="1276 558 1787 899"> <p>Containers</p> <ul style="list-style-type: none"> Containers include the app and all of its dependencies, but <u>share the kernel</u> with other containers. Run as an isolated process in userspace on the host operating system. <u>Not</u> tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. </div> </div> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Container Cloud Services</p> <p>The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container.</p> <p>To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming.</p> <p>To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services.</p> <p>Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p> <p>Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p>https://www.techtarget.com/searchdatacenter/definition/kernel</p>

Claim 1	Accused Instrumentalities
<p>[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;</p>	<p>The method practiced by Oracle and/or its customer through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers.</p> <p>For example, OCI and/or OKE stores application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent storage available to each node running the application. The terms “node” and “host” are both used to refer to the claimed server. The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network. In addition to the application software, each container includes associated system files, including a Linux user space required to execute the application, for example libc/glibc and other shared libraries, configuration files, etc. necessary for the application. For example, the container includes a base OS image provided by Oracle or by a third party. The container is compatible with the host kernel, for example because the container libraries are linked against the Linux kernel, and the supported host operating systems also use the Linux kernel, which has a stable binary interface.</p> <p>The containers are secure containers as claimed. For example, the data within an individual container is insulated from the effects of other containers except to the extent the container is specifically configured to allow other containers to modify its data, for example using a shared volume.</p> <p><i>See, e.g.:</i></p> <h2 data-bbox="730 1019 1312 1076">Overview of File Storage</h2> <p>Oracle Cloud Infrastructure File Storage service provides a durable, scalable, secure, enterprise-grade network file system. You can connect to a File Storage service file system from any bare metal, virtual machine, or container instance in your Virtual Cloud Network (VCN). You can also access a file system from outside the VCN using VCN peering, Oracle Cloud Infrastructure FastConnect, and Internet Protocol security (IPSec) virtual private network (VPN).</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="722 266 1031 310">What is Docker?</h2> <hr data-bbox="722 347 783 354"/> <p data-bbox="722 410 1902 558">A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly.</p> <p data-bbox="722 589 1917 675">Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle.</p> <p data-bbox="722 706 1860 792">Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical <a data-bbox="1045 737 1234 760" href="#">open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products.</p> <p data-bbox="722 816 1677 846"><a data-bbox="722 816 1677 846" href="https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/">https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

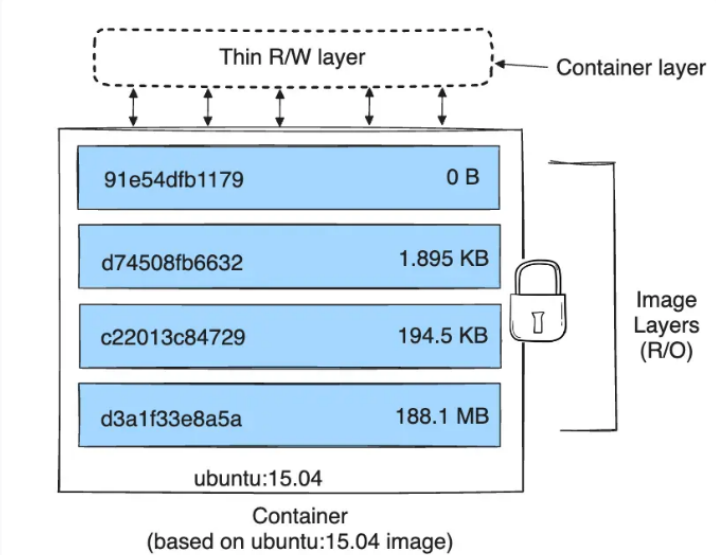
Claim 1	Accused Instrumentalities
	<p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>Docker engine:</p> <p>The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu.</p> <p>Docker images:</p> <p>Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image.</p> <p>Docker Registry:</p> <p>Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p> <p>Docker versus Kubernetes</p> <p>Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p>  <div style="display: flex; justify-content: space-around;"> <div data-bbox="861 933 1228 1015"> <p>Virtual Machines</p> <ul style="list-style-type: none"> Each virtual machine (VM) includes the app, the necessary binaries and libraries and an <u>entire guest operating system</u> </div> <div data-bbox="1281 933 1669 1096"> <p>Containers</p> <ul style="list-style-type: none"> Containers include the app and all of its dependencies, but <u>share the kernel</u> with other containers. Run as an isolated process in userspace on the host operating system. <u>Not</u> tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. </div> </div> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 267 844 300">Docker</p> <p data-bbox="730 332 999 360">Images and Containers</p> <p data-bbox="730 389 1921 451">An image is a read-only template with instructions for creating a Docker container and an image is based on another image.</p> <p data-bbox="730 483 1843 545">A container is a standard unit of software that packages up code and all its dependencies. Hence, the application runs quickly and reliably from one environment to another.</p> <p data-bbox="730 574 1911 636">A Docker Container Image is a lightweight, standalone, executable package of software that includes everything needed to run an application such as code, runtime, system tools, system libraries, and settings.</p> <p data-bbox="730 665 1915 834">Container images become containers at runtime and for Docker containers, the images become containers when they run on the engine. Containers are available for both Linux and Windows-based applications. The containerized software always runs the same code, regardless of the infrastructure. The container isolates software from its environment and ensures that it works uniformly despite differences for instance between Development, Staging, and Production.</p> <p data-bbox="730 876 995 915">Kubernetes (K8)</p> <p data-bbox="730 945 1866 1042">Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. It groups the containers that makes an application into logical units for easy management and discovery.</p> <p data-bbox="714 1071 1554 1136">https://docs.oracle.com/en/industries/financial-services/microservices-common/14.6.1.0.0/contg/technologies.html</p> <p data-bbox="743 1172 1037 1214">Container images</p> <p data-bbox="743 1243 1304 1360">A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p data-bbox="714 1380 1272 1412">https://kubernetes.io/docs/concepts/containers/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="722 261 1440 282">6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p data-bbox="722 318 1684 396">Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.</p> <p data-bbox="722 410 1808 436">https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p> <h2 data-bbox="722 477 1312 540">About storage drivers</h2> <p data-bbox="722 586 1873 699">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="722 764 1585 816">Storage drivers versus Docker volumes</h2> <p data-bbox="722 854 1911 1101">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="722 1151 1904 1265">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="722 1297 1268 1323">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="732 266 1129 318">Images and layers</h2> <p data-bbox="732 354 1827 423">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="751 492 1478 781"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="732 846 1898 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="714 1154 1266 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer structure of a Docker container. At the bottom is the base image, labeled 'ubuntu:15.04'. Above it are four stacked layers, each with a unique hash and size: '91e54dfb1179' (0 B), 'd74508fb6632' (1.895 KB), 'c22013c84729' (194.5 KB), and 'd3a1f33e8a5a' (188.1 MB). These are collectively labeled 'Image Layers (R/O)' with a padlock icon. On top of these is a 'Thin R/W layer', which is also labeled 'Container layer' with an arrow. Bidirectional arrows connect the container layer to the image layers below it. Below the diagram, it says 'Container (based on ubuntu:15.04 image)'.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2>Volumes</h2> <p>Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p>https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2>Container environment</h2> <p>The Kubernetes Container environment provides several important resources to Containers:</p> <ul style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p>https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 940 331">Images</h2> <p data-bbox="737 363 1545 505">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="737 540 1551 607">You typically create a container image of your application and push it to a registry before referring to it in a <code>Pod</code>.</p> <p data-bbox="714 633 1365 662">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="730 703 980 761">Volumes</h2> <p data-bbox="730 795 1554 1211">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <code>volume</code> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="714 1237 1344 1266">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="747 277 1335 331"><h2>Open Container Initiative</h2></div> <div data-bbox="747 391 1226 435"><h3>Image Format Specification</h3></div> <div data-bbox="747 479 1890 548"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="747 581 1900 651"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="711 678 1503 743"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 1	Accused Instrumentalities
	<p data-bbox="726 272 896 310">Overview</p> <p data-bbox="726 362 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="743 623 1906 980"> <p>The diagram illustrates the components of an image manifest. On the left, a code block for a <code>HelloWorld</code> class is shown. An arrow points from this code to a cylinder labeled 'layer' containing the paths <code>/bin/java</code>, <code>/opt/app.jar</code>, and <code>/lib/libc</code>. To the right of the layer is a plus sign, followed by a document icon labeled 'image index' containing a JSON snippet: <code>{ "manifests": { "platform": { "os": "linux", ...</code>. Another plus sign follows, leading to a document icon labeled 'config' containing a JSON snippet: <code>{ ... "config": { "Cmd": ["java", "-jar", "app.jar"], ...</code>. Each of the three components (layer, image index, and config) has a small square icon with the letters 'ci' in the top-left corner.</p> </div> <p data-bbox="714 1008 1503 1073">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 266 1335 321">OCI Image Configuration</h2> <p data-bbox="730 370 1913 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="730 558 1671 589">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="714 621 1528 686">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="737 272 821 306">Layer</p> <ul data-bbox="764 342 1915 667" style="list-style-type: none">• Image filesystems are composed of <i>layers</i>.• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="737 716 924 750">Image JSON</p> <ul data-bbox="764 786 1915 1110" style="list-style-type: none">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.• This JSON is considered to be immutable, because changing it would change the computed ImageID.• Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="714 1136 1528 1200">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

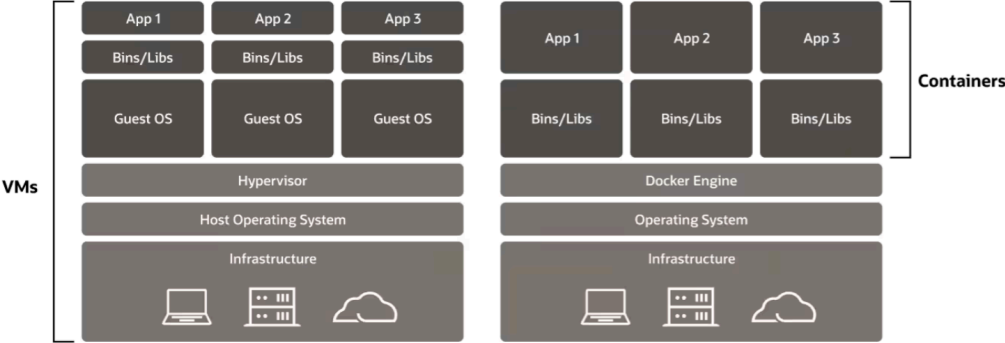
Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none">• rootfs object, REQUIRED<p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p><ul style="list-style-type: none">◦ type string, REQUIRED<p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p>◦ diff_ids array of strings, REQUIRED<p>An array of layer content hashes (<code>DiffIDs</code>), in order from first to last.</p><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
<p>[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,</p>	<p>In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems.</p> <p>The associated system files in the container are compatible with the host kernel, for example because they are linked against the Linux kernel and the supported host operating systems also use the Linux kernel, which has a stable binary interface.</p> <p><i>See</i> discussion in element [1a] above.</p> <p><i>See also, e.g.:</i></p> <p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
<p>[1c] the containers of application software excluding a kernel,</p>	<p>In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the containers of application software exclude a kernel.</p> <p><i>See, e.g.:</i></p> <p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p> <p>6. Do Docker containers package up the entire OS and make it easier to deploy?</p> <p>Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers.</p> <p>https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/</p>

Claim 1	Accused Instrumentalities
<p>[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server.</p> <p>For example, each container will utilize its own associated system files, including libraries such as libc/glibc and configuration files, not the corresponding associated local system files (<i>e.g.</i>, libraries and configuration files of the host OS). As described above and below, in the Accused Instrumentalities the associated system files provide at least some of the same functionalities as the associated local system files. The host/node's associated local system files remain resident on the host/node, for example for use by system processes or applications outside the container environment.</p> <p><i>See, e.g.:</i></p> <p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>Docker engine:</p> <p>The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu.</p> <p>Docker images:</p> <p>Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image.</p> <p>Docker Registry:</p> <p>Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

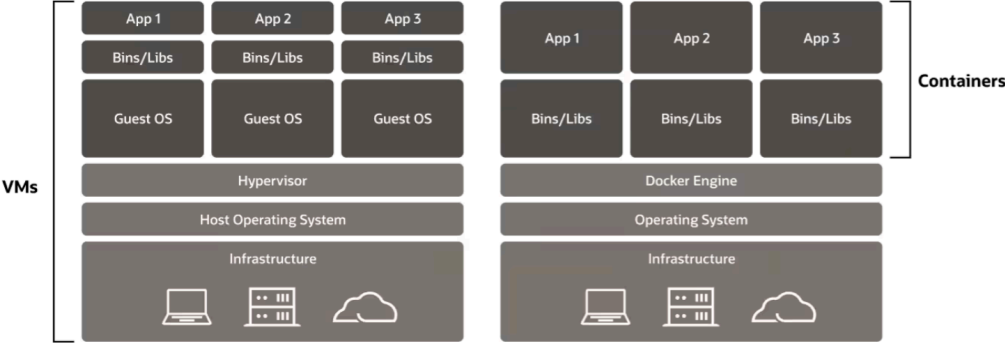
Claim 1	Accused Instrumentalities
	<p data-bbox="724 267 1087 297">Docker versus Kubernetes</p> <p data-bbox="724 324 1915 597">Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers.</p> <p data-bbox="714 630 1677 659">https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p>  <div style="display: flex; justify-content: space-around;"> <div data-bbox="861 933 1218 1015"> <p>Virtual Machines</p> <ul style="list-style-type: none"> Each virtual machine (VM) includes the app, the necessary binaries and libraries and an <u>entire guest operating system</u> </div> <div data-bbox="1281 933 1659 1096"> <p>Containers</p> <ul style="list-style-type: none"> Containers include the app and all of its dependencies, but <u>share the kernel</u> with other containers. Run as an isolated process in userspace on the host operating system. <u>Not</u> tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. </div> </div> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 269 1083 297">Container Cloud Services</p> <p data-bbox="730 326 1902 380">The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container.</p> <p data-bbox="730 412 1927 651">To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming.</p> <p data-bbox="730 683 1913 862">To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services.</p> <p data-bbox="730 894 1923 980">Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications.</p> <p data-bbox="714 1013 1680 1045">https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
<p>[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,</p>	<p>In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server.</p> <p>For example, in some cases the host OS and container will use one or more identical system files, for example when both the host and the container incorporate the same Linux distribution version, or when both host and container use the same version of libc. In the case where the associated system files are identical to the associated local system files, they are copies thereof. In other cases modified copies are used instead, for example when different versions of the same library, or configuration files with different parameters, are used by the host and container.</p> <p><i>See, e.g.:</i></p> <p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>Docker engine:</p> <p>The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu.</p> <p>Docker images:</p> <p>Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image.</p> <p>Docker Registry:</p> <p>Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="722 266 1087 297">Docker versus Kubernetes</p> <p data-bbox="722 323 1919 594">Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers.</p> <p data-bbox="722 626 1677 657">https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

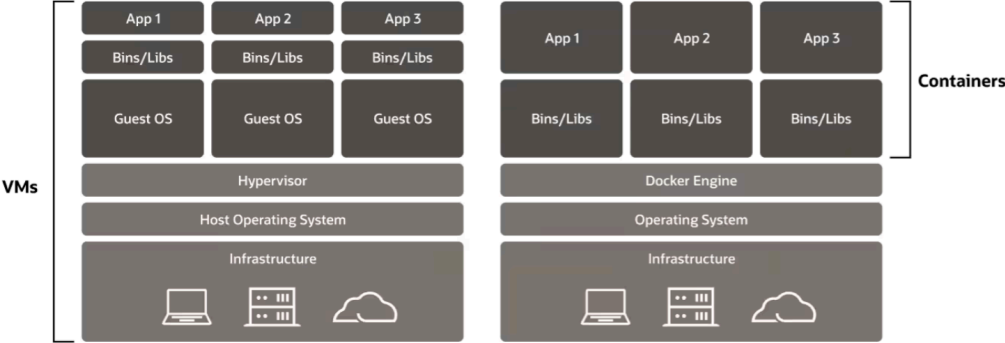
Claim 1	Accused Instrumentalities
	<p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p>  <p>Virtual Machines</p> <ul style="list-style-type: none"> • Each virtual machine (VM) includes the app, the necessary binaries and libraries and an <u>entire guest operating system</u> <p>Containers</p> <ul style="list-style-type: none"> • Containers include the app and all of its dependencies, but <u>share the kernel</u> with other containers. • Run as an isolated process in userspace on the host operating system. • <u>Not</u> tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Container Cloud Services</p> <p>The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container.</p> <p>To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming.</p> <p>To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services.</p> <p>Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p> <p><code>COPY</code> and <code>ADD</code> : These commands copy files and directories from your local filesystem into the Docker image. They are often used to include your application code, configuration files, and dependencies.</p> <p>https://medium.com/@swalperen3008/what-is-dockerize-and-dockerize-your-project-a-step-by-step-guide-899c48a34df6</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="743 264 1037 310">Container images</p> <p data-bbox="743 337 1304 456">A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p data-bbox="714 496 1270 526">https://kubernetes.io/docs/concepts/containers/</p>

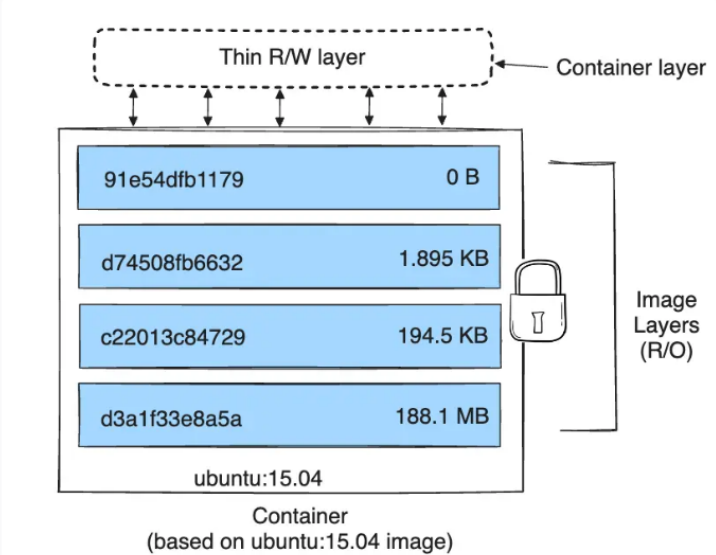
Claim 1	Accused Instrumentalities
<p>[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,</p>	<p>In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software.</p> <p>For example, each container has an isolated runtime environment that cannot be accessed by other containers, for example including a per-container writeable layer or other ephemeral per-container storage. For another example, when the plurality of secure containers each corresponds to a different container image, each container cannot access another container's image and therefore application software.</p> <p><i>See, e.g.:</i></p> <p>Cgroups and Namespaces History</p> <p>The underlying Linux kernel features that Docker uses are cgroups and namespaces. In 2008 cgroups were introduced to the Linux kernel based on work previously done by Google developers¹. Cgroups limit and account for the resource usage of a set of operating system processes.</p> <p>The Linux kernel uses namespace to isolate the system resources of processes from each other. The first namespace, i.e. the mount namespace, was introduced as early as 2002.²</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>Docker engine:</p> <p>The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu.</p> <p>Docker images:</p> <p>Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image.</p> <p>Docker Registry:</p> <p>Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p> <p>Docker versus Kubernetes</p> <p>Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p>  <p>Virtual Machines</p> <ul style="list-style-type: none"> • Each virtual machine (VM) includes the app, the necessary binaries and libraries and an <u>entire guest operating system</u> <p>Containers</p> <ul style="list-style-type: none"> • Containers include the app and all of its dependencies, but <u>share the kernel</u> with other containers. • Run as an isolated process in userspace on the host operating system. • <u>Not</u> tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="722 269 1312 331">About storage drivers</h2> <p data-bbox="722 376 1871 493">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="722 558 1583 610">Storage drivers versus Docker volumes</h2> <p data-bbox="722 646 1913 896">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="722 941 1902 1058">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1373 987 1556 1011" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="722 1088 1266 1117"><a data-bbox="722 1088 1266 1117" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="735 266 1129 318">Images and layers</h2> <p data-bbox="735 354 1827 425">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="735 464 1906 805"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="735 844 1906 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="735 1153 1268 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer architecture of a container. At the bottom is a stack of four blue rectangular blocks representing image layers (Read-Only). From top to bottom, they are labeled with their commit IDs and sizes: <code>91e54dfb1179</code> (0 B), <code>d74508fb6632</code> (1.895 KB), <code>c22013c84729</code> (194.5 KB), and <code>d3a1f33e8a5a</code> (188.1 MB). To the right of this stack is a padlock icon and the text "Image Layers (R/O)". Above this stack is a dashed-line box labeled "Thin R/W layer". Arrows point from the top of the image layers stack to the thin R/W layer. To the right of the thin R/W layer is an arrow pointing to it from the text "Container layer". Below the entire stack is the text "Container (based on ubuntu:15.04 image)".</p> <p>https://docs.docker.com/storage/storagedriver/</p>

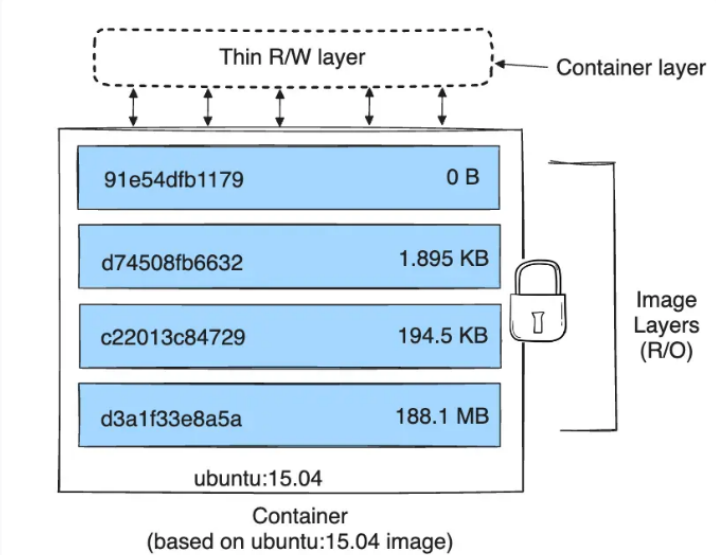
Claim 1	Accused Instrumentalities
<p>[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.</p>	<p>In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system.</p> <p>For example, the container's root file system comprises the image layer(s), an ephemeral writeable layer (e.g., in Docker terminology the container layer), and optionally one or more volumes. This root file system is distinct and isolated from the host operating system's root file system.</p> <p><i>See, e.g.:</i></p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="747 272 1740 326">Setting Up Storage for Kubernetes Clusters</h2> <p data-bbox="747 365 1913 500"><i>Find out how to define and apply persistent volume claims to clusters you've created using Kubernetes Engine (OKE). With Oracle Cloud Infrastructure as the underlying IaaS provider, you can provision persistent volume claims by attaching volumes from the Block Volume service or by mounting file systems from the File Storage service.</i></p> <p data-bbox="747 540 1877 639">Container storage via a container's root file system is ephemeral, and can disappear upon container deletion and creation. To provide a durable location to prevent data from being lost, you can create and use persistent volumes to store data outside of containers.</p> <p data-bbox="747 680 1829 743">A persistent volume offers persistent storage that enables your data to remain intact, regardless of whether the containers to which the storage is connected are terminated.</p> <p data-bbox="747 784 1892 847">A persistent volume claim (PVC) is a request for storage, which is met by binding the PVC to a persistent volume (PV). A PVC provides an abstraction layer to the underlying storage.</p> <p data-bbox="747 888 1593 914">With Oracle Cloud Infrastructure, you can provision persistent volume claims:</p> <ul data-bbox="772 954 1885 1289" style="list-style-type: none">• By attaching volumes from the Oracle Cloud Infrastructure Block Volume service. The volumes are connected to clusters created by Kubernetes Engine using CSI (Container Storage Interface) or FlexVolume volume plugins deployed on the clusters. Oracle recommends the CSI volume plugin since the upstream Kubernetes project deprecates the FlexVolume volume plugin in Kubernetes version 1.23. See Provisioning PVCs on the Block Volume Service.• By mounting file systems in the Oracle Cloud Infrastructure File Storage service. The File Storage service file systems are mounted inside containers running on clusters created by Kubernetes Engine using a CSI (Container Storage Interface) volume plugin deployed on the clusters. See Provisioning PVCs on the File Storage Service. <p data-bbox="716 1300 1612 1362">https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengcreatingpersistentvolumeclaim.htm</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 269 1184 310">Configuring Docker Storage</p> <p data-bbox="730 334 1923 472">The Docker Engine is configured to use <code>overlay2</code> as the default storage driver to manage Docker containers. This provides a performance and scalability improvement on earlier releases that used the device mapper as the default storage driver, but the technology is new and should be tested properly before use in production environments. For more information on <code>overlay2</code>, see:</p> <p data-bbox="730 501 1556 532">https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/</p> <p data-bbox="730 558 1860 621">Overlay file systems can corrupt when used in conjunction with any file system that does not have <code>dtype</code> support enabled.</p> <p data-bbox="730 646 1593 709">https://docs.oracle.com/en/operating-systems/oracle-linux/docker/docker-InstallingOracleContainerRuntimeforDocker.html</p> <p data-bbox="730 738 1906 922">In Kubernetes, each container can read and write to its own file system. But when a container is restarted, all data is lost. Therefore, containers that need to maintain state would store data in a persistent storage such as Network File System (NFS). What's already stored in NFS isn't deleted when a pod, which might contain one or more containers, is destroyed. Also, an NFS can be accessed from multiple pods at the same time, so an NFS can be used to share data between pods. This behavior is really useful when containers or applications need to read configuration data from a single shared file system or when multiple containers need to read from and write data to a single shared file system.</p> <p data-bbox="730 951 1932 1170">Oracle Cloud Infrastructure File Storage provides a durable, scalable, and distributed enterprise-grade network file system that supports NFS version 3 along with Network Lock Manager (NLM) for a locking mechanism. You can connect to File Storage from any bare metal, virtual machine, or container instance in your virtual cloud network (VCN). You can also access a file system from outside the VCN by using Oracle Cloud Infrastructure FastConnect or an Internet Protocol Security (IPSec) virtual private network (VPN). File Storage is a fully managed service so you don't have to worry about hardware installations and maintenance, capacity planning, software upgrades, security patches, and so on. You can start with a file system that contains only a few kilobytes of data and grows to handle 8 exabytes of data.</p> <p data-bbox="730 1182 1900 1245">https://blogs.oracle.com/cloud-infrastructure/post/using-file-storage-service-with-container-engine-for-kubernetes</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="722 269 1310 334">About storage drivers</h2> <p data-bbox="722 376 1871 493">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="722 558 1583 610">Storage drivers versus Docker volumes</h2> <p data-bbox="722 646 1913 896">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="722 941 1902 1058">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the <a data-bbox="1373 987 1556 1011" href="#">volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="722 1088 1268 1117"><a data-bbox="722 1088 1268 1117" href="https://docs.docker.com/storage/storagedriver/">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="735 267 1129 316">Images and layers</h2> <p data-bbox="735 354 1827 423">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="735 467 1480 803"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="735 846 1900 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="735 1154 1270 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer architecture of a Docker container. At the bottom is the base image, labeled 'ubuntu:15.04'. Above it is a stack of four image layers, each represented by a blue box with a hash and a size: '91e54dfb1179' (0 B), 'd74508fb6632' (1.895 KB), 'c22013c84729' (194.5 KB), and 'd3a1f33e8a5a' (188.1 MB). A padlock icon and the label 'Image Layers (R/O)' indicate these are read-only. On top of the stack is a dashed box labeled 'Thin R/W layer', which is also labeled 'Container layer' with an arrow. Arrows show the relationship between the container layer and the image layers below it.</p> <p>Container (based on ubuntu:15.04 image)</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 277 982 337">Volumes</h2> <p data-bbox="730 386 1906 509">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="714 529 1344 561">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="730 607 1268 656">Container environment</h2> <p data-bbox="730 688 1499 753">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="772 786 1478 938" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="714 971 1549 1003">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="737 272 940 331">Images</h2> <p data-bbox="737 363 1545 505">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="737 540 1551 607">You typically create a container image of your application and push it to a registry before referring to it in a <code>Pod</code>.</p> <p data-bbox="714 633 1365 662">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="730 703 980 761">Volumes</h2> <p data-bbox="730 795 1554 1211">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <code>volume</code> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="714 1237 1344 1266">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="747 277 1335 331"><h2>Open Container Initiative</h2></div> <div data-bbox="747 391 1226 435"><h3>Image Format Specification</h3></div> <div data-bbox="747 479 1890 548"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="747 581 1900 651"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="711 678 1505 743"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

Claim 1	Accused Instrumentalities
	<p data-bbox="728 272 896 310">Overview</p> <p data-bbox="728 362 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="743 623 1906 980"> <p>The diagram illustrates the components of an image manifest. On the left, a code block for a 'HelloWorld' class is shown. An arrow points from this code to a cylinder labeled 'layer' containing the paths '/bin/java', '/opt/app.jar', and '/lib/libc'. This layer is then combined with an 'image index' (a document icon) which contains a JSON structure: { "manifests": { "platform": { "os": "linux", ... } } }. This is further combined with a 'config' (another document icon) which contains a JSON structure: { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } }. Each component (layer, image index, and config) has a small square icon with the letters 'ci' in the top-left corner.</p> </div> <p data-bbox="714 1008 1503 1073">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>

Claim 1	Accused Instrumentalities
	<h2>OCI Image Configuration</h2> <p>An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p>This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="741 274 821 306">Layer</p> <ul data-bbox="766 342 1913 667" style="list-style-type: none">• Image filesystems are composed of <i>layers</i>.• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="741 716 924 748">Image JSON</p> <ul data-bbox="766 784 1913 1109" style="list-style-type: none">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.• This JSON is considered to be immutable, because changing it would change the computed ImageID.• Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="716 1138 1526 1203">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<ul style="list-style-type: none">• rootfs <i>object</i>, REQUIRED<p>The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash.</p><ul style="list-style-type: none">◦ type <i>string</i>, REQUIRED<p>MUST be set to <code>layers</code>. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image.</p>◦ diff_ids <i>array of strings</i>, REQUIRED<p>An array of layer content hashes (<code>DiffIDs</code>), in order from first to last.</p><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Exhibit 3



US007784058B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.:** **US 7,784,058 B2**
(45) **Date of Patent:** **Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES**

2002/0004854 A1 1/2002 Hartley
2002/0174215 A1 11/2002 Schaefer 709/224
2003/0101292 A1 5/2003 Fisher
2004/0025165 A1* 2/2004 Desoli et al. 719/310

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul
O’Leary**, Kanata (CA); **Dean Huffman**,
Kanata (CA)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002
WO WO 2006/039181 A 4/2006

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

OTHER PUBLICATIONS

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 1293 days.

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

* cited by examiner

(21) Appl. No.: **10/946,536**

Primary Examiner—Hyung S Sough

Assistant Examiner—Syed Roni

(22) Filed: **Sep. 21, 2004**

(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt,
Milbrath & Gilchrist, P.A. Attorneys at Law

(65) **Prior Publication Data**
US 2005/0066303 A1 Mar. 24, 2005

(57) **ABSTRACT**

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep.
22, 2003.

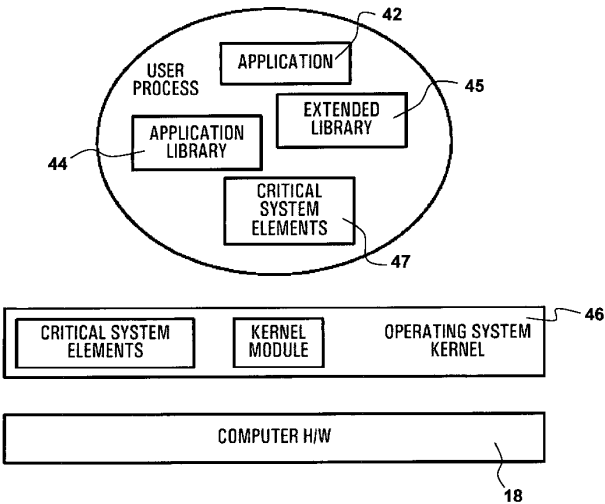
A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

(51) **Int. Cl.**
G06F 9/22 (2006.01)
(52) **U.S. Cl.** **719/310**; 719/319
(58) **Field of Classification Search** 719/310,
719/319
See application file for complete search history.

(56) **References Cited**
U.S. PATENT DOCUMENTS

5,481,706 A * 1/1996 Peek 710/200
6,212,574 B1 * 4/2001 O’Rourke et al. 719/321
6,260,075 B1 * 7/2001 Cabrero et al. 719/310

18 Claims, 7 Drawing Sheets



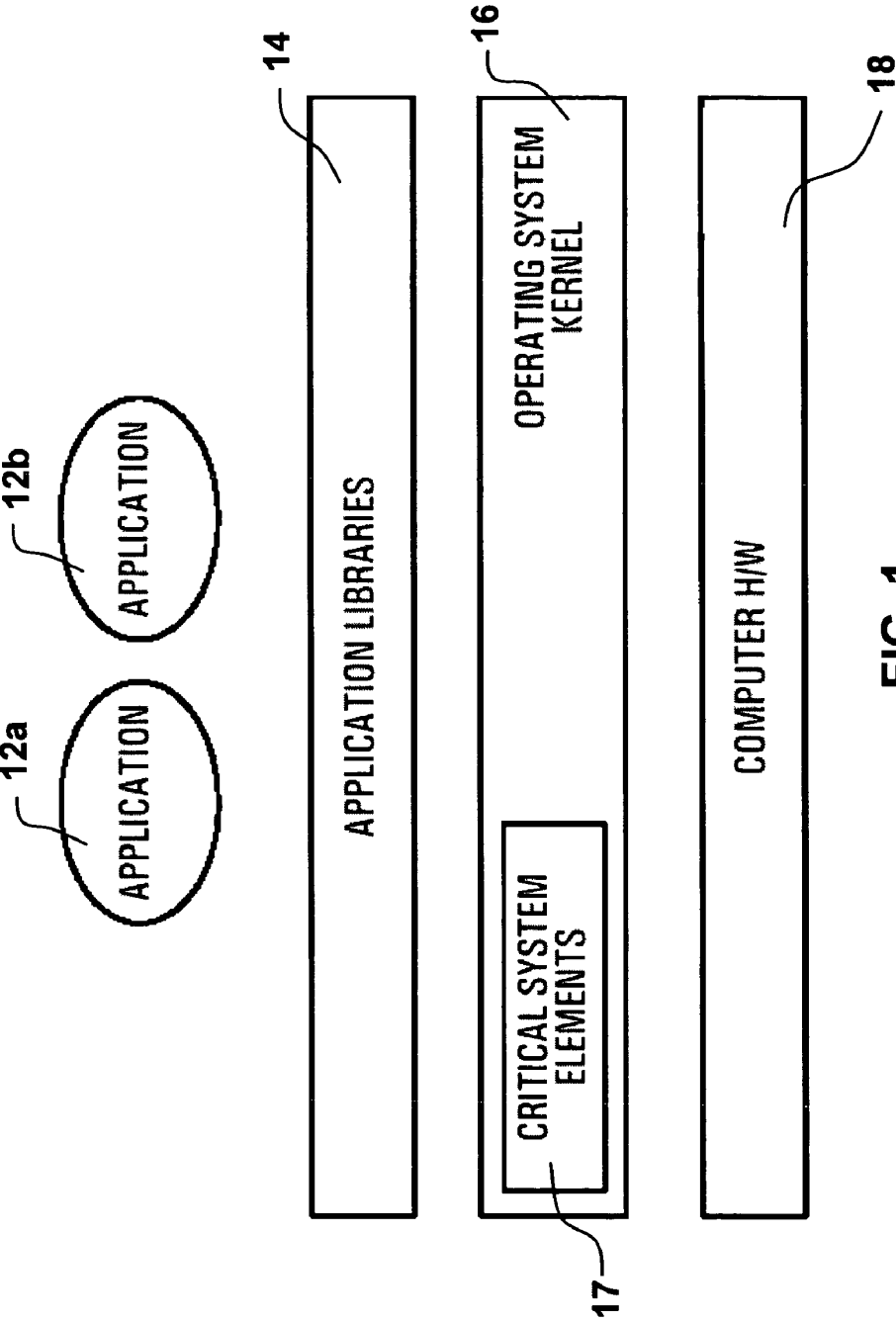


FIG. 1
Prior Art

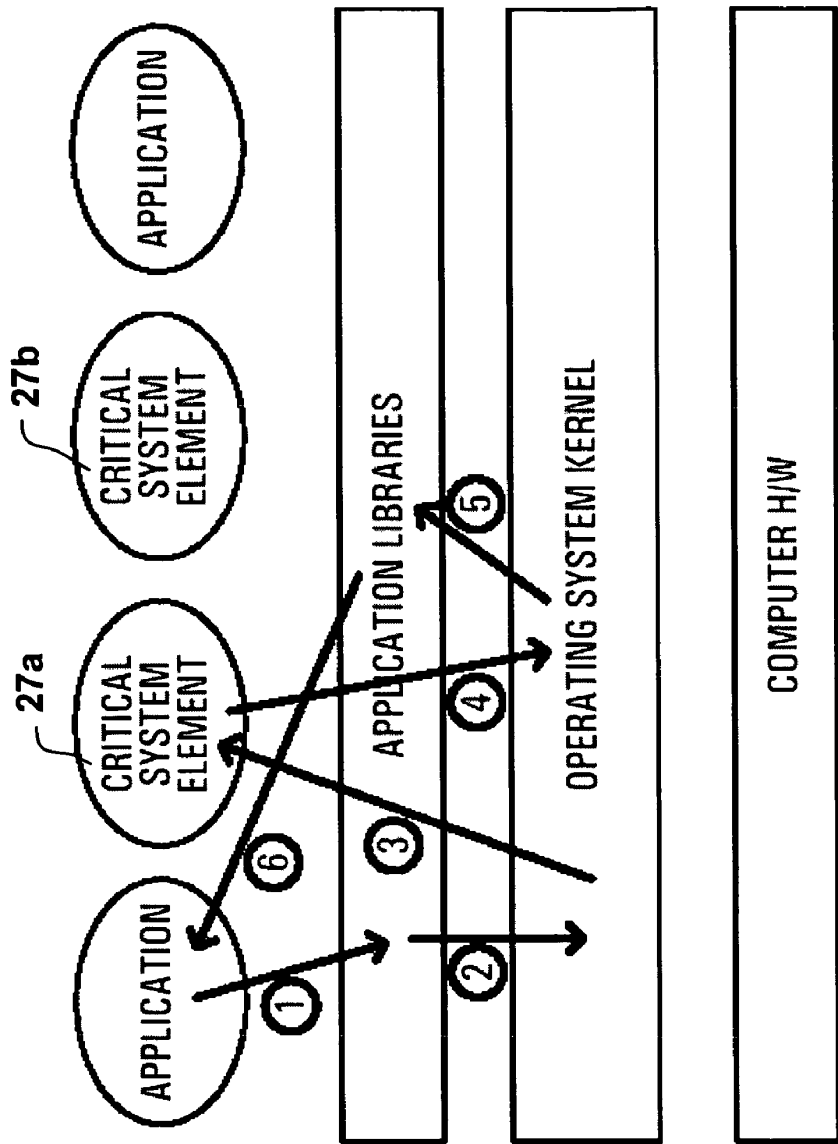


FIG. 2a
Prior Art

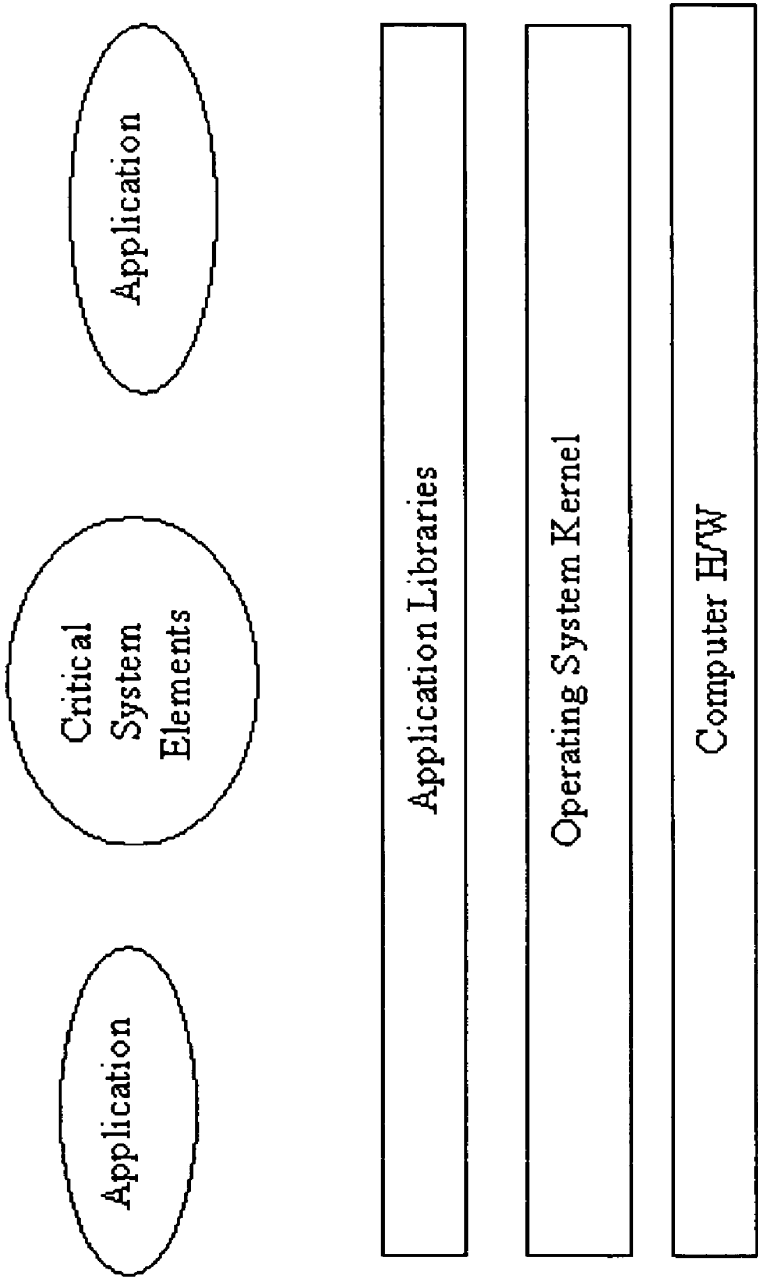


FIG. 2b
Prior Art

U.S. Patent

Aug. 24, 2010

Sheet 4 of 7

US 7,784,058 B2

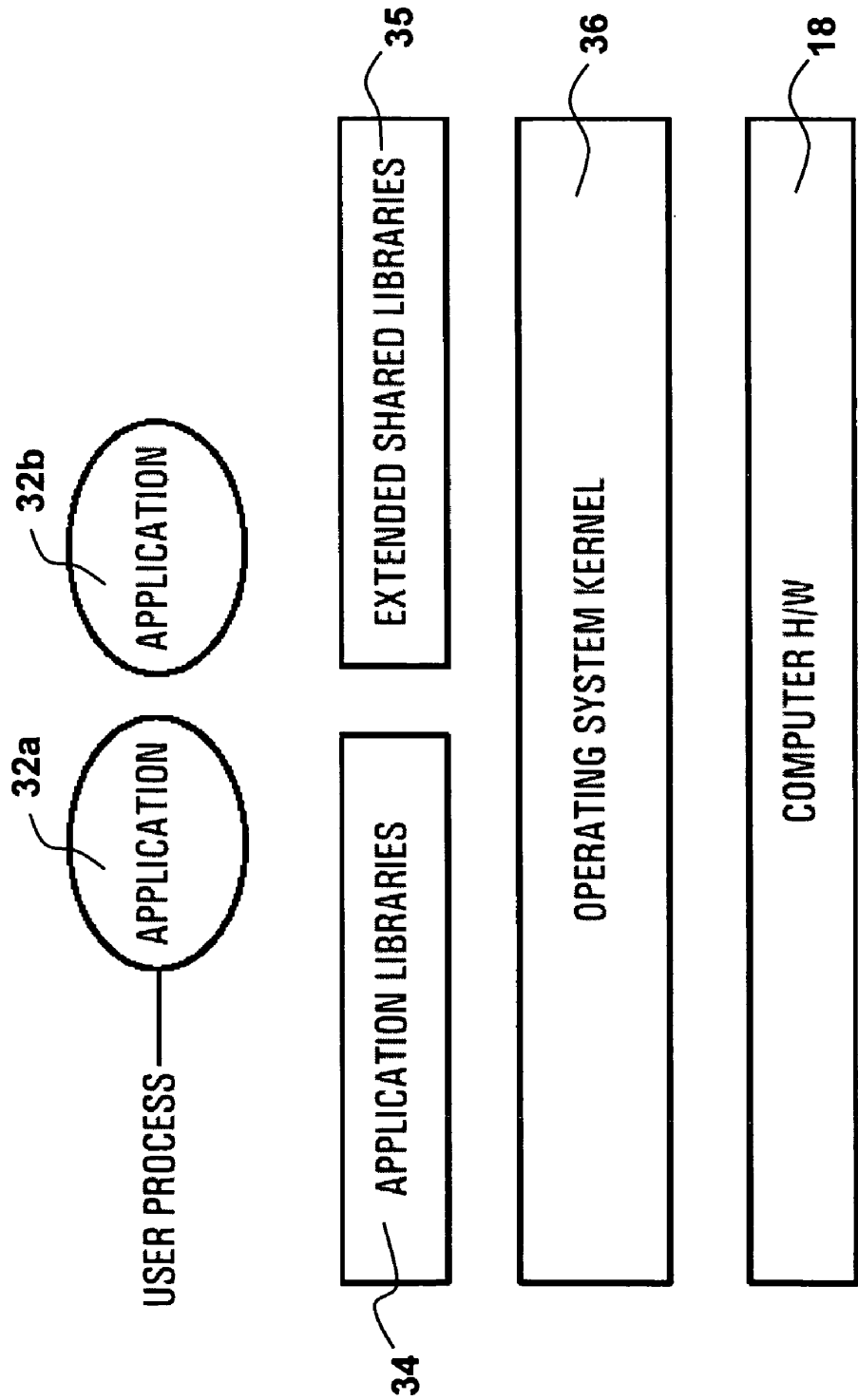
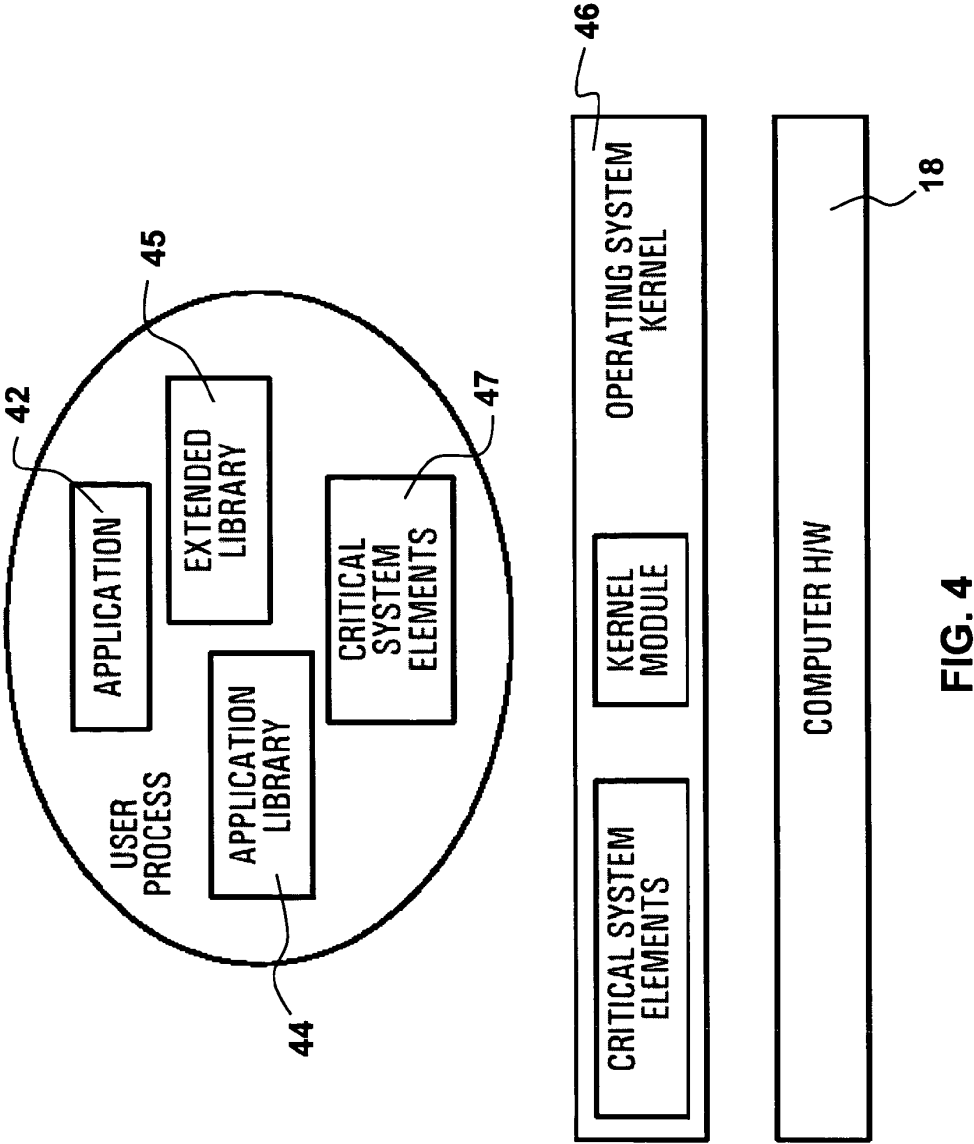


FIG. 3



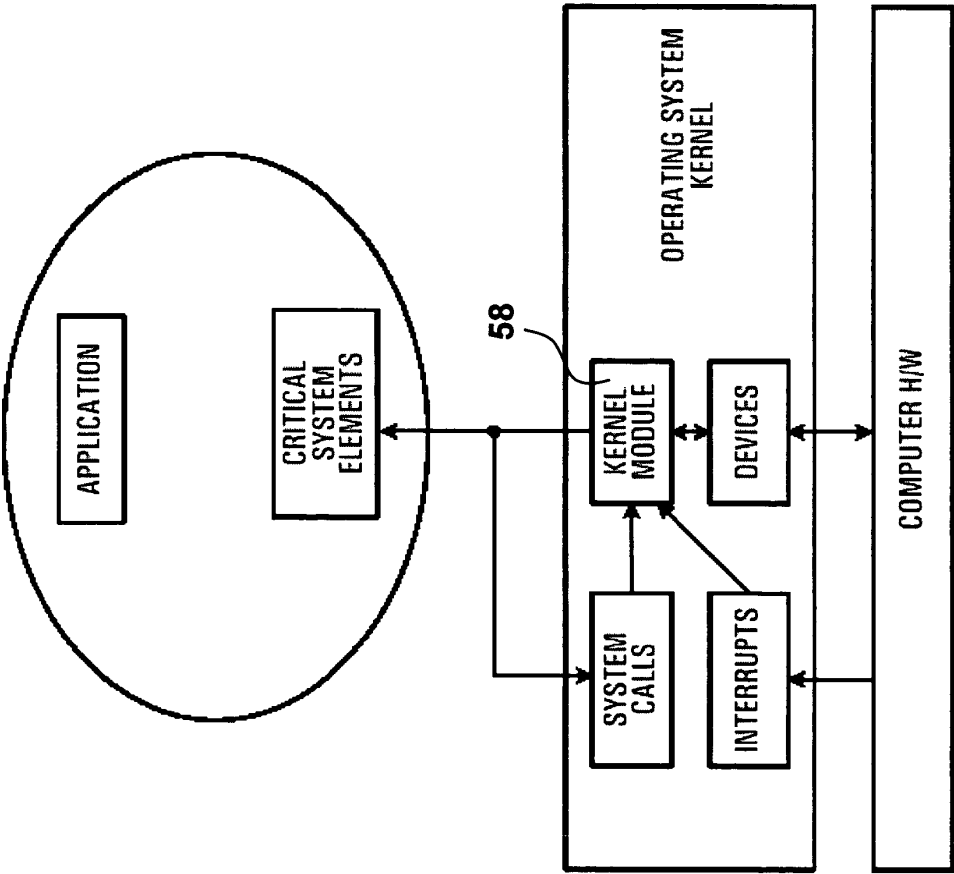


FIG. 5

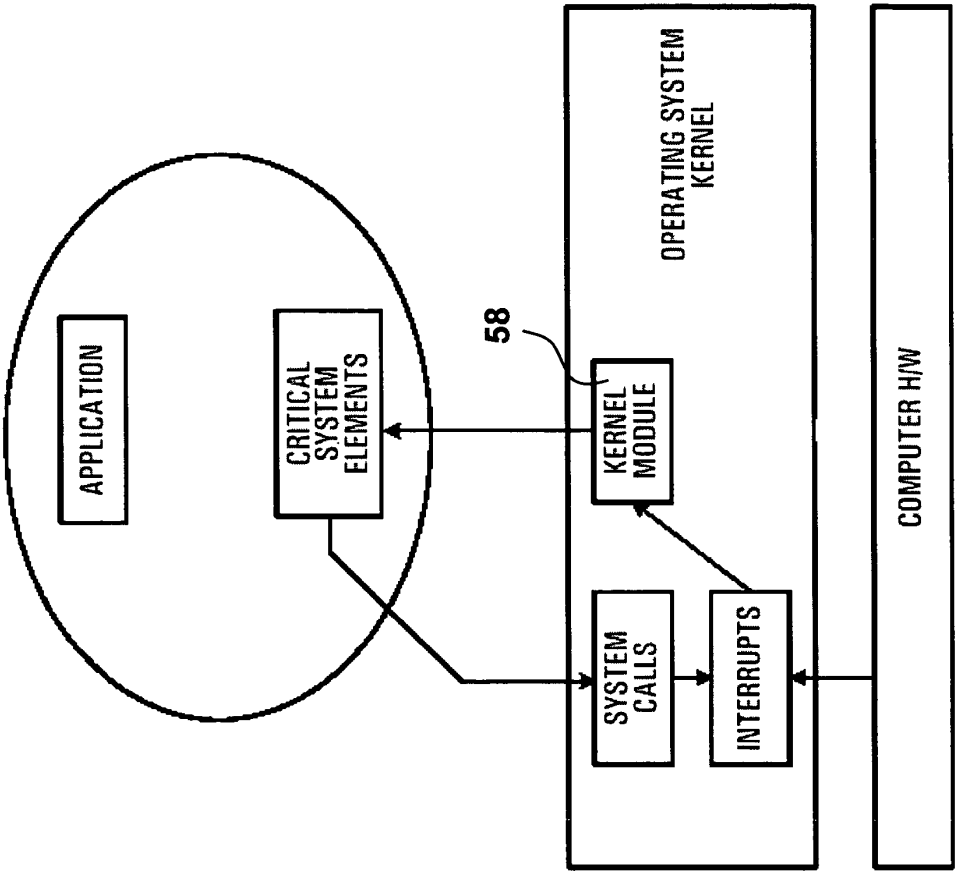


FIG. 6

US 7,784,058 B2

1

**COMPUTING SYSTEM HAVING USER MODE
CRITICAL SYSTEM ELEMENTS AS SHARED
LIBRARIES****CROSS-REFERENCE TO RELATED
APPLICATIONS**

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled “User Mode Critical System Element as Shared Libs”, which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

2

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.

2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

US 7,784,058 B2

3

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

4

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. 1 is an architectural view of the traditional monolithic prior art operating system;

US 7,784,058 B2

5

FIG. 2a is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. 2b is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. 3 is an architectural view of an embodiment of the invention;

FIG. 4 is a functional view showing how critical system elements exist in the same context as an application;

FIG. 5 is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. 6 shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TCP/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP

6

stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality

2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided

2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. 1 shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. 1 to make use of a critical system element 17 in the kernel 16, the application 12a or 12b

US 7,784,058 B2

7

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12a**, **12b** makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12a**, **12b** in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2a** shows a system architecture where critical system elements **27a**, **27b** execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2a** is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2a** as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2b** is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2a** and FIG. **2b** is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2b** is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the

8

case with a multi-server architecture depicted in FIG. **2a**. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32a**, **32b**, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45,47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

US 7,784,058 B2

9

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. 3 shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. 4 shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. 4 the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. 4. FIG. 5 shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

10

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. 6 a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:

- a) a processor;
- b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,
- c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and
 - i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,
 - ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

US 7,784,058 B2

11

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

2. A computing system as defined in claim 1, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim 1 wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim 1 wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

6. A computing system according to claim 5 wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

7. A computing system according to claim 6 wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

8. A computing system according to claim 7 wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

12

9. A computing system according to claim 2, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

10. A computing system according to claim 2 wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

11. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

12. A computing system according to claim 1, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

13. A computing system according to claim 10 wherein some SLCSEs are modified for a particular one of the plurality of software applications.

14. A computing system according to claim 13 wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

15. A computing system according to claim 5 wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

16. A computing system according to claim 1 wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

17. A computing system according to claim 2 wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

18. A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs.

* * * * *

Exhibit 4

U.S. Patent No. 7,784,058 vs. Oracle

Accused Instrumentalities: Oracle products and services using user mode critical system elements as shared libraries, including without limitation Oracle Cloud Infrastructure (“OCI”) and Oracle Kubernetes Engine (“OKE”), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
[1pre] 1. A computing system for executing a plurality of software applications comprising:	To the extent the preamble is limiting, each Accused Instrumentality comprises or constitutes a computing system for executing a plurality of software applications as claimed. <i>See claim limitations below.</i> <i>See also, e.g.:</i>

Claim 1	Accused Instrumentalities
	<div data-bbox="730 269 1108 318"> <h2>Why Choose OKE?</h2> </div> <div data-bbox="846 386 989 496"> </div> <div data-bbox="730 540 917 565"> <p>Price-Performance</p> </div> <div data-bbox="730 589 1092 680"> <p>OKE is the lowest cost Kubernetes service amongst all hyperscalers, especially for serverless.</p> </div> <div data-bbox="1266 386 1388 496"> </div> <div data-bbox="1140 540 1262 565"> <p>Autoscaling</p> </div> <div data-bbox="1140 589 1516 680"> <p>OKE automatically adjusts compute resources based on demand, which can reduce your costs.</p> </div> <div data-bbox="1692 386 1785 496"> </div> <div data-bbox="1545 540 1652 565"> <p>Efficiency</p> </div> <div data-bbox="1545 589 1904 680"> <p>GPUs can be scarce, but OKE job scheduling makes it easy to maximize resource utilization.</p> </div> <div data-bbox="850 748 997 833"> </div> <div data-bbox="730 891 840 915"> <p>Portability</p> </div> <div data-bbox="730 940 1108 1031"> <p>OKE is consistent across clouds and on-premises, enabling portability and avoiding vendor lock-in.</p> </div> <div data-bbox="1251 737 1392 846"> </div> <div data-bbox="1140 891 1245 915"> <p>Simplicity</p> </div> <div data-bbox="1140 940 1520 1031"> <p>OKE reduces the time and cost needed to manage the complexities of Kubernetes infrastructure.</p> </div> <div data-bbox="1686 742 1785 846"> </div> <div data-bbox="1545 891 1654 915"> <p>Reliability</p> </div> <div data-bbox="1545 940 1913 1031"> <p>Automatic upgrades and security patching boost reliability for the control plane and worker nodes.</p> </div> <div data-bbox="709 1075 1470 1105"> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/</p> </div>

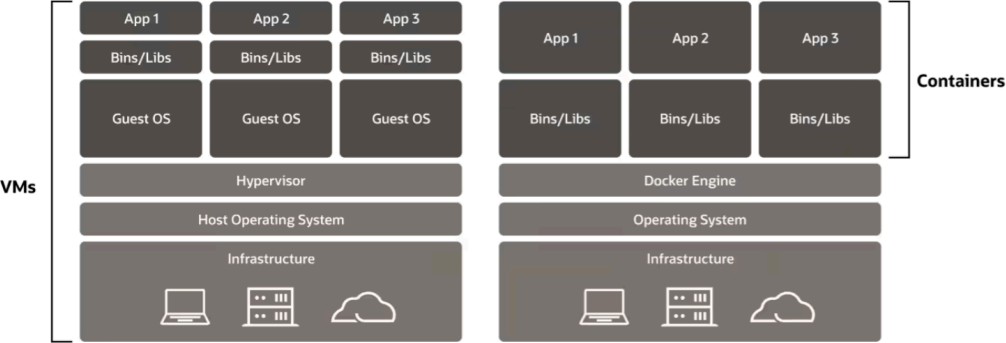
Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 274 1671 326">Welcome to Oracle Cloud Infrastructure</h2> <p data-bbox="730 370 1902 509">Oracle Cloud Infrastructure (OCI) is a set of complementary cloud services that enable you to build and run a range of applications and services in a highly available hosted environment. OCI provides high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network.</p> <p data-bbox="714 532 1642 558">https://docs.oracle.com/en-us/iaas/Content/GSG/Concepts/baremetalintro.htm</p> <h3 data-bbox="758 618 1835 721">Existing applications can benefit by migrating to OCI and OKE</h3> <p data-bbox="758 769 1766 802">OKE offers lower total cost of ownership and improved time to market.</p> <p data-bbox="758 862 1556 894">OKE simplifies operations at scale in the following ways:</p> <ul data-bbox="758 980 1835 1333" style="list-style-type: none"><li data-bbox="758 980 1251 1057">– Lift and shift; there's no need to rearchitect<li data-bbox="758 1117 1251 1193">– Reduce operations burden with automation<li data-bbox="758 1253 1251 1330">– Save time on infrastructure management<li data-bbox="1318 980 1818 1057">– Increase resource utilization and efficiency<li data-bbox="1318 1117 1835 1193">– Improve agility, flexibility, uptime, and resilience<li data-bbox="1318 1253 1755 1330">– Reduce compliance risk and enhance security <p data-bbox="714 1377 1650 1403">https://www.oracle.com/cloud/cloud-native/kubernetes-engine/#app-migration</p>

Claim 1	Accused Instrumentalities
	<p>What is OCI Kubernetes Engine (OKE)?</p> <p>Oracle Cloud Infrastructure Kubernetes Engine (OKE) is a managed Kubernetes service that simplifies the development, deployment, and operation of containerized workloads at scale. OKE enables you to quickly create, manage, and consume Kubernetes clusters that leverage underlying OCI compute, networking, and storage services.</p> <p>When should I use OKE?</p> <p>You should use OKE when you want to leverage Kubernetes to deploy and manage your Kubernetes-based container applications. It allows you to combine the production-grade container orchestration of standard upstream Kubernetes with the control, security, and high, predictable performance of OCI.</p> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p> <p>How does OKE provide resiliency?</p> <p>When you create an OKE cluster, OKE automatically creates and manages multiple Kubernetes control plane nodes spread across fault domains and availability domains (logical data centers). This is done to help ensure that the managed Kubernetes control plane is highly available. Control plane operations, such as upgrading to newer versions of Kubernetes, can be performed without service interruptions. Additionally, when you provision worker nodes, you can use a placement configuration to control the fault domain and availability domain where they are created. Nodes will automatically come online with labels, which you can use to schedule your workloads so they are robust and highly available.</p> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p> <p>Can I deploy private Kubernetes clusters?</p> <p>Yes; with OKE, your Kubernetes clusters are integrated in your VCN. Your cluster worker nodes, load balancers, and the Kubernetes API endpoint are part of a private or public subnet of your VCN. Regular VCN routing and firewall rules control access to the Kubernetes API endpoint, making it accessible from a corporate network only, through a bastion host, or by specific platform services.</p> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p>

Claim 1	Accused Instrumentalities
	<p>When should I use virtual nodes, managed nodes, or self-managed nodes?</p> <ul style="list-style-type: none"> Virtual nodes Virtual nodes offer a serverless Kubernetes experience. This option is ideal if you'd rather focus on your application and avoid managing the underlying infrastructure. Virtual nodes relieve you of management-related tasks such as scaling, upgrading, patching, troubleshooting, and provisioning worker nodes. Managed nodes Managed nodes are a good choice for general purpose workloads. They offer an extensive list of customizable configuration options that have been tested by the OKE service. Unlike fully managed virtual nodes, you share the management of worker nodes with OCI. OKE simplifies the management process through features such as on-demand cycling to automate worker node updates, cluster self-healing upon failure detection, autoscaling, and more. Self-managed nodes Self-managed nodes offer access to the underlying infrastructure, configuration options, and compute shapes that aren't currently available to managed nodes. This includes access to specialized infrastructure, such as RDMA-enabled bare metal cluster networks or confidential compute shapes. This advanced control makes self-managed nodes ideal for specialized use cases that aren't supported with managed nodes. Note that with self-managed nodes, you are fully responsible for managing the worker nodes—without the automated features provided by managed or virtual nodes. <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p> <p>What are the storage options for virtual nodes?</p> <p>OKE virtual nodes do not yet have persistent storage capabilities. However, there are plans to introduce support for attaching persistent volumes backed by OCI Block Storage and OCI File Storage. If your Kubernetes application requires persistent storage, it's advisable to use OKE managed nodes.</p> <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="747 272 1562 321">Supported Images for Managed Nodes</h2> <p data-bbox="747 363 1917 427">Kubernetes Engine supports the provisioning of worker nodes (managed nodes only) using some, but not all, of the latest Oracle Linux images provided by Oracle Cloud Infrastructure.</p> <p data-bbox="747 469 1581 493">You can use these Oracle Linux images when provisioning managed nodes:</p> <ul data-bbox="772 535 991 667" style="list-style-type: none"><li data-bbox="772 535 945 560">• <a data-bbox="810 535 945 560" href="#">OKE Images<li data-bbox="772 586 991 610">• <a data-bbox="810 586 991 610" href="#">Platform Images<li data-bbox="772 636 982 660">• <a data-bbox="810 636 982 660" href="#">Custom Images <p data-bbox="711 703 1785 727"><a data-bbox="711 703 1785 727" href="https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengimagesshapes.htm">https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengimagesshapes.htm</p> <h2 data-bbox="722 769 1031 810">What is Docker?</h2> <p data-bbox="722 911 1902 1057">A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly.</p> <p data-bbox="722 1089 1917 1175">Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle.</p> <p data-bbox="722 1208 1860 1294">Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical <a data-bbox="835 1240 961 1265" href="#">open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products.</p> <p data-bbox="711 1320 1677 1344"><a data-bbox="711 1320 1677 1344" href="https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/">https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>Docker engine:</p> <p>The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu.</p> <p>Docker images:</p> <p>Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image.</p> <p>Docker Registry:</p> <p>Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p> <p>Docker versus Kubernetes</p> <p>Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p>  <div style="display: flex; justify-content: space-around;"> <div data-bbox="861 933 1218 1015"> <p>Virtual Machines</p> <ul style="list-style-type: none"> • Each virtual machine (VM) includes the app, the necessary binaries and libraries and an <u>entire guest operating system</u> </div> <div data-bbox="1281 933 1659 1096"> <p>Containers</p> <ul style="list-style-type: none"> • Containers include the app and all of its dependencies, but <u>share the kernel</u> with other containers. • Run as an isolated process in userspace on the host operating system. • <u>Not</u> tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. </div> </div> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>


Claim 1	Accused Instrumentalities
	<p data-bbox="730 267 1081 297">Container Cloud Services</p> <p data-bbox="730 326 1902 380">The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container.</p> <p data-bbox="730 409 1927 651">To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming.</p> <p data-bbox="730 680 1911 862">To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services.</p> <p data-bbox="730 891 1923 979">Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications.</p> <p data-bbox="714 1015 1677 1044">https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

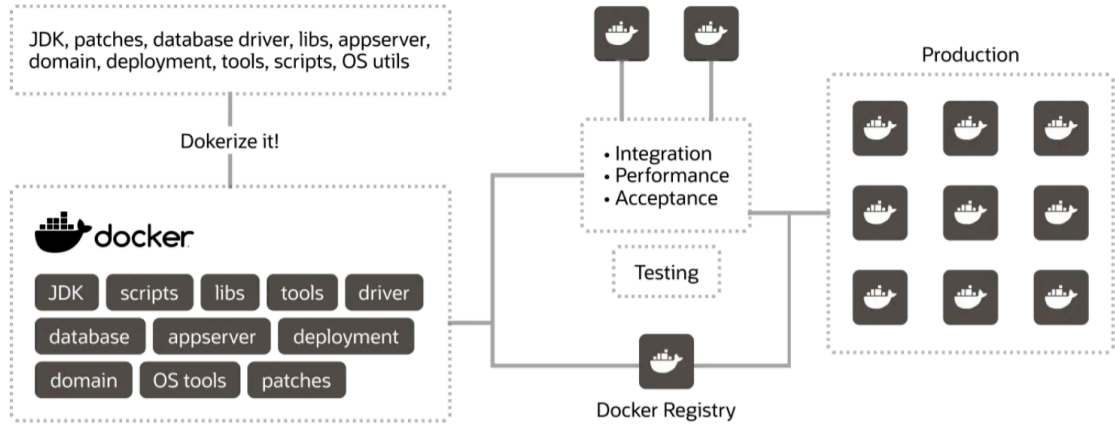
Claim 1	Accused Instrumentalities
<p>[1a] a) a processor;</p>	<p>Each Accused Instrumentality comprises a processor.</p> <p>For example, each node/host contains at least one CPU.</p> <p><i>See, e.g.:</i></p> <p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p> <div data-bbox="777 714 1785 1055"> </div> <div data-bbox="840 1088 1218 1185"> <p>Virtual Machines</p> <ul style="list-style-type: none"> • Each virtual machine (VM) includes the app, the necessary binaries and libraries and an <u>entire guest operating system</u> </div> <div data-bbox="1260 1088 1680 1266"> <p>Containers</p> <ul style="list-style-type: none"> • Containers include the app and all of its dependencies, but <u>share the kernel</u> with other containers. • Run as an isolated process in userspace on the host operating system. • <u>Not</u> tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. </div> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>When should I use virtual nodes, managed nodes, or self-managed nodes?</p> <ul style="list-style-type: none"> Virtual nodes Virtual nodes offer a serverless Kubernetes experience. This option is ideal if you'd rather focus on your application and avoid managing the underlying infrastructure. Virtual nodes relieve you of management-related tasks such as scaling, upgrading, patching, troubleshooting, and provisioning worker nodes. Managed nodes Managed nodes are a good choice for general purpose workloads. They offer an extensive list of customizable configuration options that have been tested by the OKE service. Unlike fully managed virtual nodes, you share the management of worker nodes with OCI. OKE simplifies the management process through features such as on-demand cycling to automate worker node updates, cluster self-healing upon failure detection, autoscaling, and more. Self-managed nodes Self-managed nodes offer access to the underlying infrastructure, configuration options, and compute shapes that aren't currently available to managed nodes. This includes access to specialized infrastructure, such as RDMA-enabled bare metal cluster networks or confidential compute shapes. This advanced control makes self-managed nodes ideal for specialized use cases that aren't supported with managed nodes. Note that with self-managed nodes, you are fully responsible for managing the worker nodes—without the automated features provided by managed or virtual nodes. <p>https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/</p> <h2>Welcome to Oracle Cloud Infrastructure</h2> <p>Oracle Cloud Infrastructure (OCI) is a set of complementary cloud services that enable you to build and run a range of applications and services in a highly available hosted environment. OCI provides high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network.</p> <p>https://docs.oracle.com/en-us/iaas/Content/GSG/Concepts/baremetalintro.htm</p>

Claim 1	Accused Instrumentalities
<p>[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,</p>	<p>Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor.</p> <p>For example, the OSCSEs include kernel-mode functions similar to the functionalities provided by user-space libraries such as glibc. These are implemented in kernel-space to handle tasks such as (without limitation) memory management (kmalloc(), kfree(), etc.) at kernel level.</p> <p><i>See, e.g.:</i></p> <p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p> <div data-bbox="777 821 1785 1164"> </div> <div data-bbox="858 1200 1203 1282"> <p>Virtual Machines</p> <ul style="list-style-type: none"> Each virtual machine (VM) includes the app, the necessary binaries and libraries and an entire guest operating system </div> <div data-bbox="1276 1200 1659 1362"> <p>Containers</p> <ul style="list-style-type: none"> Containers include the app and all of its dependencies, but share the kernel with other containers. Run as an isolated process in userspace on the host operating system. Not tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. </div> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Kernel mode</p> <p>Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode.</p> <p>https://www.techtarget.com/searchdatacenter/definition/kernel</p> <p>The GNU C Library, commonly known as glibc, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system.</p> <p>https://en.wikipedia.org/wiki/Glibc</p>

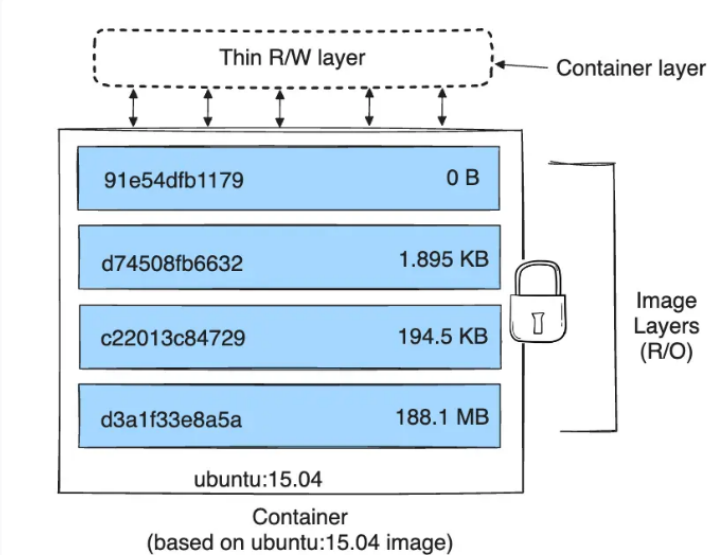
Claim 1	Accused Instrumentalities
<p>[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and</p>	<p>Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode.</p> <p>For example, the shared library with SLCSEs include the runtime environment, system tools, and dependencies, such as the glibc library and other libraries that replicate OSCSEs, included in the container image (including without limitation in a base image that is included within the container image).</p> <p><i>See, e.g.:</i></p>  <p>Figure 2</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Docker Image—Development to Production</p> <p>Creating a Docker image with all of its dependencies solves the "but it worked for me on my development machine" problem. The key idea is that a Docker image is created automatically by a build pipeline from a source-code repository like Git and initially tested in a development environment. This immutable image will then be stored in a Docker registry.</p> <p>As shown in the Figure 4, the same image will be used for further load tests, integration tests, acceptance tests, and more. In every environment, the same image will be used. Small but necessary environmentally specific differences, such as a JDBC URL for a production database, can be fed into the container as environment variables or files.</p>  <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="747 266 1087 310">Container images</p> <p data-bbox="747 334 1398 451">A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p data-bbox="716 469 1272 500">https://kubernetes.io/docs/concepts/containers/</p> <p data-bbox="716 521 1562 695">Container image files are complete, static and executable versions of an application or service and differ from one technology to another. Docker images are made up of multiple layers, which start with a base image that includes all of the dependencies needed to execute code in a container. Each image has a readable/writable layer on top of static unchanging layers. Because each container has its own specific container layer that customizes that specific container, underlying image layers can be saved and reused in multiple containers. An Open Container Initiative (OCI)</p> <p data-bbox="716 704 1881 768">https://www.techtarget.com/searchitoperations/definition/container-containerization-or-container-based-virtualization</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="722 269 1310 331">About storage drivers</h2> <p data-bbox="722 376 1871 493">To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2 data-bbox="722 558 1583 610">Storage drivers versus Docker volumes</h2> <p data-bbox="722 646 1913 896">Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p data-bbox="722 941 1902 1058">Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.</p> <p data-bbox="722 1088 1268 1117">https://docs.docker.com/storage/storagedriver/</p>

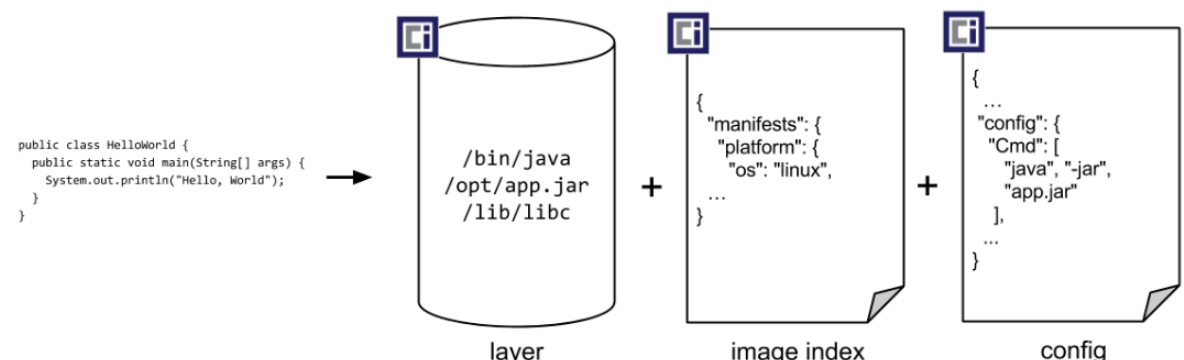
Claim 1	Accused Instrumentalities
	<h2 data-bbox="732 266 1131 315">Images and layers</h2> <p data-bbox="732 354 1827 423">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="751 488 1480 781"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="732 846 1898 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="714 1154 1268 1182">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer structure of a container. At the bottom, a box labeled 'Container (based on ubuntu:15.04 image)' contains a stack of four 'Image Layers (R/O)'. Each layer is represented by a blue rectangle with a hash and a size: '91e54dfb1179' (0 B), 'd74508fb6632' (1.895 KB), 'c22013c84729' (194.5 KB), and 'd3a1f33e8a5a' (188.1 MB). A padlock icon is shown next to the stack, indicating they are read-only. Above this stack is a dashed box labeled 'Thin R/W layer', which is identified as the 'Container layer' by an arrow. Bidirectional arrows connect the container layer to the top of the image layers stack.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 277 982 337">Volumes</h2> <p data-bbox="730 386 1906 508">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="714 532 1344 561">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="730 607 1268 654">Container environment</h2> <p data-bbox="730 688 1501 751">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="772 786 1478 938" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="714 971 1551 1000">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 272 940 331">Images</h2> <p data-bbox="730 363 1545 506">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="730 539 1551 607">You typically create a container image of your application and push it to a registry before referring to it in a <code>Pod</code>.</p> <p data-bbox="730 633 1365 662">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="730 704 978 760">Volumes</h2> <p data-bbox="730 795 1554 1211">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <code>volume</code> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="730 1237 1344 1266">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="749 277 1335 331">Open Container Initiative</h2> <hr data-bbox="749 344 1906 347"/> <h3 data-bbox="749 391 1226 435">Image Format Specification</h3> <hr data-bbox="749 444 1906 448"/> <p data-bbox="749 480 1906 548">This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p> <p data-bbox="749 581 1906 649">The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p> <p data-bbox="711 678 1505 743">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>





Claim 1	Accused Instrumentalities
	<p>Overview</p> <p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p>  <pre> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World!"); } } </pre> <p>→</p> <p>layer: /bin/java, /opt/app.jar, /lib/libc</p> <p>image index: { "manifests": { "platform": { "os": "linux", ... } ... }</p> <p>config: { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } }</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 266 1335 321">OCI Image Configuration</h2> <p data-bbox="730 370 1913 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="730 558 1671 589">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="714 621 1528 686">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="737 272 821 305">Layer</p> <ul data-bbox="764 342 1915 667" style="list-style-type: none">• Image filesystems are composed of <i>layers</i>.• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="737 716 924 748">Image JSON</p> <ul data-bbox="764 786 1915 1110" style="list-style-type: none">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.• This JSON is considered to be immutable, because changing it would change the computed ImageID.• Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="711 1138 1528 1198">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

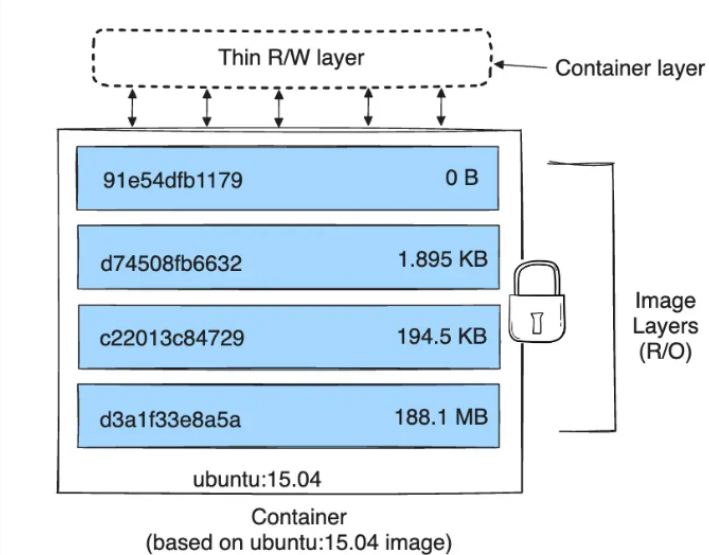
Claim 1	Accused Instrumentalities
	<p>The GNU C Library, commonly known as glibc, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system.</p> <p>https://en.wikipedia.org/wiki/Glibc</p>

Claim 1	Accused Instrumentalities
<p>[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,</p>	<p>In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications.</p> <p>For example, a base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and dependencies, including system libraries or shared library critical system elements (SLCSEs). The base image forms a part of the container image according to the “layer” model described in the documentation below. When the container runs the image, it creates a runtime instance of that container image. In turn, when one or more applications executes within the container runtime environment, it dynamically links to the SLCSEs stored in the runtime environment, which thereby become a part of the application(s).</p> <p><i>See, e.g.:</i></p> <p>Container images</p> <p>A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings.</p> <p>https://kubernetes.io/docs/concepts/containers/</p>

Claim 1	Accused Instrumentalities																																																						
	<div>  <div> ubuntu  </div> <div> Updated 15 days ago Ubuntu is a Debian-based Linux operating system based on free software. </div> <div> Linux IBM Z 386 riscv64 x86-64 ARM ARM 64 PowerPC 64 LE </div> </div> <div> <div>  <div> debian  </div> <div> Updated 35 minutes ago Debian is a Linux distribution that's composed entirely of free and open-source software. </div> <div> Linux riscv64 x86-64 ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z </div> </div> <p>https://hub.docker.com/search?image_filter=official&type=image&q=</p> <table border="1"> <thead> <tr> <th>Platform</th> <th>x86_64 / amd64</th> <th>arm64 / aarch64</th> <th>arm (32-bit)</th> <th>ppc64le</th> <th>s390x</th> </tr> </thead> <tbody> <tr> <td>CentOS</td> <td>✓</td> <td>✓</td> <td></td> <td>✓</td> <td></td> </tr> <tr> <td>Debian</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td></td> </tr> <tr> <td>Fedora</td> <td>✓</td> <td>✓</td> <td></td> <td>✓</td> <td></td> </tr> <tr> <td>Raspberry Pi OS (32-bit)</td> <td></td> <td></td> <td>✓</td> <td></td> <td></td> </tr> <tr> <td>RHEL (s390x)</td> <td></td> <td></td> <td></td> <td></td> <td>✓</td> </tr> <tr> <td>SLES</td> <td></td> <td></td> <td></td> <td></td> <td>✓</td> </tr> <tr> <td>Ubuntu</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> </tr> <tr> <td>Binaries</td> <td>✓</td> <td>✓</td> <td>✓</td> <td></td> <td></td> </tr> </tbody> </table> <p>https://docs.docker.com/engine/install/</p> </div>	Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x	CentOS	✓	✓		✓		Debian	✓	✓	✓	✓		Fedora	✓	✓		✓		Raspberry Pi OS (32-bit)			✓			RHEL (s390x)					✓	SLES					✓	Ubuntu	✓	✓	✓	✓	✓	Binaries	✓	✓	✓		
Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x																																																		
CentOS	✓	✓		✓																																																			
Debian	✓	✓	✓	✓																																																			
Fedora	✓	✓		✓																																																			
Raspberry Pi OS (32-bit)			✓																																																				
RHEL (s390x)					✓																																																		
SLES					✓																																																		
Ubuntu	✓	✓	✓	✓	✓																																																		
Binaries	✓	✓	✓																																																				

Claim 1	Accused Instrumentalities
	<p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <h2>About storage drivers</h2> <p>To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way.</p> <h2>Storage drivers versus Docker volumes</h2> <p>Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer.</p> <p>Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance.</p> <p>https://docs.docker.com/storage/storagedriver/</p>

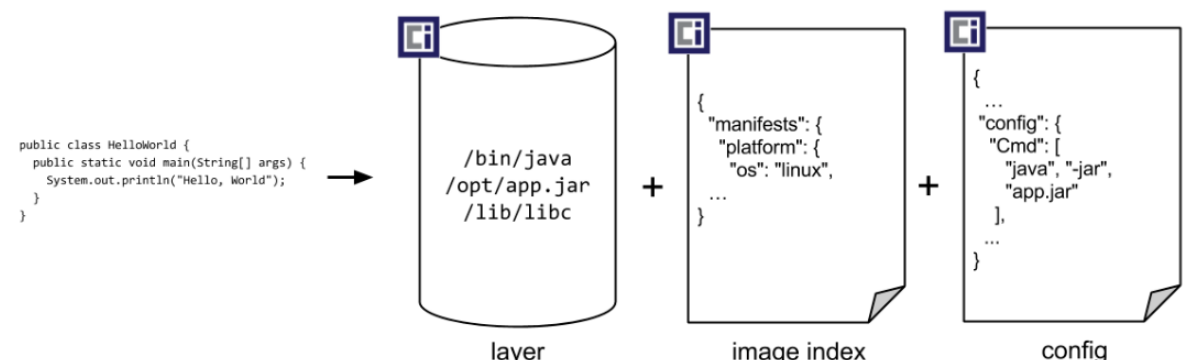
Claim 1	Accused Instrumentalities
	<h2 data-bbox="735 267 1129 316">Images and layers</h2> <p data-bbox="735 354 1827 427">A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile:</p> <pre data-bbox="735 467 1480 803"># syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r \$HOME/.cache CMD python /app/app.py</pre> <p data-bbox="735 844 1900 1133">This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The <code>FROM</code> statement starts out by creating a layer from the <code>ubuntu:22.04</code> image. The <code>LABEL</code> command only modifies the image's metadata, and doesn't produce a new layer. The <code>COPY</code> command adds some files from your Docker client's current directory. The first <code>RUN</code> command builds your application using the <code>make</code> command, and writes the result to a new layer. The second <code>RUN</code> command removes a cache directory, and writes the result to a new layer. Finally, the <code>CMD</code> instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer.</p> <p data-bbox="735 1153 1270 1185">https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<p>Each layer is only a set of differences from the layer before it. Note that both <i>adding</i>, and <i>removing</i> files will result in a new layer. In the example above, the <code>\$HOME/.cache</code> directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images.</p> <p>The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an <code>ubuntu:15.04</code> image.</p>  <p>The diagram illustrates the layer architecture of a container. At the bottom is a stack of four blue rectangular blocks representing read-only image layers. From top to bottom, they are labeled with their commit IDs and sizes: <code>91e54dfb1179</code> (0 B), <code>d74508fb6632</code> (1.895 KB), <code>c22013c84729</code> (194.5 KB), and <code>d3a1f33e8a5a</code> (188.1 MB). A bracket on the right side of these blocks is labeled "Image Layers (R/O)". A padlock icon is positioned to the right of the stack, indicating they are read-only. Above this stack is a dashed-line rectangular box labeled "Thin R/W layer". A bracket on the right side of this box is labeled "Container layer". Arrows point from the top of each image layer to the bottom of the container layer, showing that the container layer is built upon the image layers. Below the entire stack, the text "ubuntu:15.04" is centered, followed by "Container (based on ubuntu:15.04 image)".</p> <p>https://docs.docker.com/storage/storagedriver/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 277 982 337">Volumes</h2> <p data-bbox="730 386 1906 508">Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts:</p> <p data-bbox="714 529 1344 561">https://kubernetes.io/docs/concepts/storage/volumes/</p> <h2 data-bbox="730 607 1268 654">Container environment</h2> <p data-bbox="730 688 1501 751">The Kubernetes Container environment provides several important resources to Containers:</p> <ul data-bbox="768 786 1478 938" style="list-style-type: none">• A filesystem, which is a combination of an image and one or more volumes.• Information about the Container itself.• Information about other objects in the cluster. <p data-bbox="714 969 1551 1002">https://kubernetes.io/docs/concepts/containers/container-environment/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="732 272 940 332">Images</h2> <p data-bbox="732 362 1545 505">A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment.</p> <p data-bbox="732 540 1551 607">You typically create a container image of your application and push it to a registry before referring to it in a <code>Pod</code>.</p> <p data-bbox="714 633 1365 662">https://kubernetes.io/docs/concepts/containers/images/</p> <h2 data-bbox="728 703 980 760">Volumes</h2> <p data-bbox="728 795 1554 1211">On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a <code>Pod</code> and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes <code>volume</code> abstraction solves both of these problems. Familiarity with <code>Pods</code> is suggested.</p> <p data-bbox="714 1237 1344 1266">https://kubernetes.io/docs/concepts/storage/volumes/</p>

Claim 1	Accused Instrumentalities
	<div data-bbox="747 272 1335 331"><h2>Open Container Initiative</h2></div> <div data-bbox="747 386 1226 435"><h3>Image Format Specification</h3></div> <div data-bbox="747 477 1890 548"><p>This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration.</p></div> <div data-bbox="747 578 1900 649"><p>The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run.</p></div> <div data-bbox="709 675 1503 740"><p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p></div>

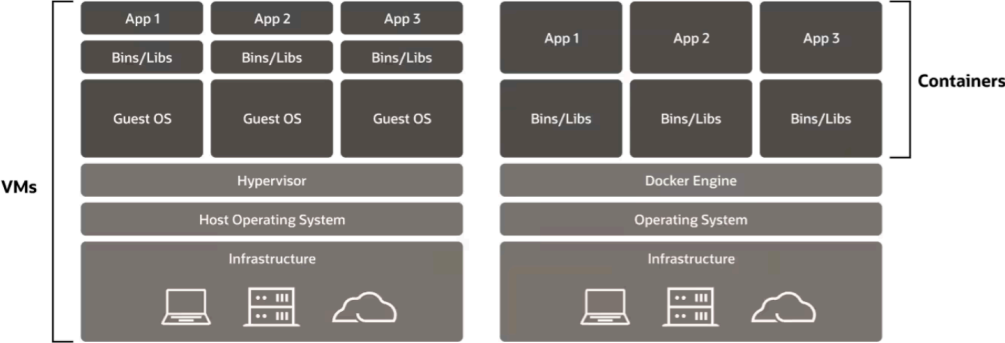
Claim 1	Accused Instrumentalities
	<p>Overview</p> <p>At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p>  <pre> public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World!"); } } </pre> <p>→</p> <p>layer: /bin/java, /opt/app.jar, /lib/libc</p> <p>image index: { "manifests": { "platform": { "os": "linux", ... } ... }</p> <p>config: { ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } }</p> <p>https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="730 266 1335 321">OCI Image Configuration</h2> <p data-bbox="730 370 1913 521">An OCI <i>Image</i> is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers.</p> <p data-bbox="730 558 1671 589">This section defines the <code>application/vnd.oci.image.config.v1+json</code> media type.</p> <p data-bbox="714 621 1528 686">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="737 272 821 305">Layer</p> <ul data-bbox="764 342 1915 667" style="list-style-type: none">• Image filesystems are composed of <i>layers</i>.• Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer.• Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer.• Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. <p data-bbox="737 716 924 748">Image JSON</p> <ul data-bbox="764 786 1915 1110" style="list-style-type: none">• Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes.• The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers.• This JSON is considered to be immutable, because changing it would change the computed ImageID.• Changing it means creating a new derived image, instead of changing the existing image. <p data-bbox="711 1138 1528 1198">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md</p> <p data-bbox="711 1226 1306 1291">Containers only have access to resources that are defined in the image, https://www.hpe.com/us/en/what-is/docker.html</p>

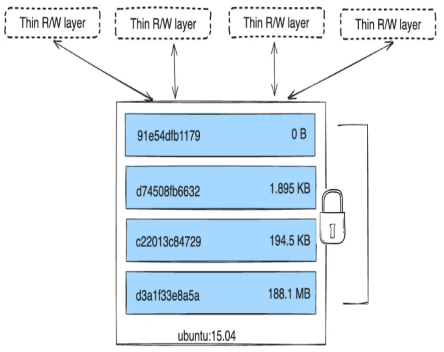
Claim 1	Accused Instrumentalities
	<p>DESCRIPTION top</p> <p>The programs ld.so and ld-linux.so* find and load the shared objects (shared libraries) needed by a program, prepare the program to run, and then run it.</p> <p>https://man7.org/linux/man-pages/man8/ld.so.8.html</p>
<p>[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and</p>	<p>In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function.</p> <p>When a Docker or Kubernetes image is used to create a container in the Accused Instrumentalities, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications running in their respective containers. The image includes essential system files, libraries, and dependencies required to run the software application within the container. The containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it is not shared with other containers. This ensures that each container has its own isolated context. Docker or Kubernetes containers in the Accused Instrumentalities can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same function.</p> <p><i>See, e.g.:</i></p>

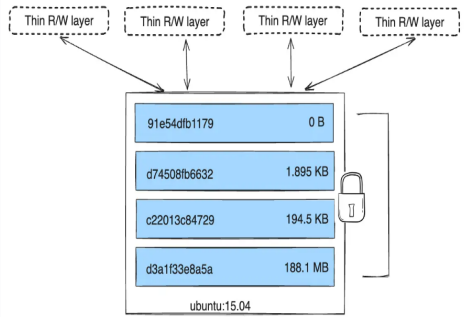
Claim 1	Accused Instrumentalities
	<p>Cgroups and Namespaces History</p> <p>The underlying Linux kernel features that Docker uses are cgroups and namespaces. In 2008 cgroups were introduced to the Linux kernel based on work previously done by Google developers¹. Cgroups limit and account for the resource usage of a set of operating system processes.</p> <p>The Linux kernel uses namespace to isolate the system resources of processes from each other. The first namespace, i.e. the mount namespace, was introduced as early as 2002.²</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p> <p>Container:</p> <p>Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the “user space.” Containers share the host system’s kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images.</p> <p>Docker engine:</p> <p>The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu.</p> <p>Docker images:</p> <p>Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image.</p> <p>Docker Registry:</p> <p>Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images.</p> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<p>Docker Basics</p> <p>The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more.</p> <p>A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.</p>  <div style="display: flex; justify-content: space-around;"> <div data-bbox="861 933 1218 1015"> <p>Virtual Machines</p> <ul style="list-style-type: none"> Each virtual machine (VM) includes the app, the necessary binaries and libraries and an <u>entire guest operating system</u> </div> <div data-bbox="1281 933 1659 1096"> <p>Containers</p> <ul style="list-style-type: none"> Containers include the app and all of its dependencies, but <u>share the kernel</u> with other containers. Run as an isolated process in userspace on the host operating system. <u>Not</u> tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. </div> </div> <p>https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/</p>

Claim 1	Accused Instrumentalities
	<h2 data-bbox="747 272 1740 326">Setting Up Storage for Kubernetes Clusters</h2> <p data-bbox="747 365 1913 500"><i>Find out how to define and apply persistent volume claims to clusters you've created using Kubernetes Engine (OKE). With Oracle Cloud Infrastructure as the underlying IaaS provider, you can provision persistent volume claims by attaching volumes from the Block Volume service or by mounting file systems from the File Storage service.</i></p> <p data-bbox="747 542 1877 639">Container storage via a container's root file system is ephemeral, and can disappear upon container deletion and creation. To provide a durable location to prevent data from being lost, you can create and use persistent volumes to store data outside of containers.</p> <p data-bbox="747 682 1829 743">A persistent volume offers persistent storage that enables your data to remain intact, regardless of whether the containers to which the storage is connected are terminated.</p> <p data-bbox="747 786 1892 847">A persistent volume claim (PVC) is a request for storage, which is met by binding the PVC to a persistent volume (PV). A PVC provides an abstraction layer to the underlying storage.</p> <p data-bbox="747 889 1593 915">With Oracle Cloud Infrastructure, you can provision persistent volume claims:</p> <ul data-bbox="772 954 1885 1289" style="list-style-type: none"> • By attaching volumes from the Oracle Cloud Infrastructure Block Volume service. The volumes are connected to clusters created by Kubernetes Engine using CSI (Container Storage Interface) or FlexVolume volume plugins deployed on the clusters. Oracle recommends the CSI volume plugin since the upstream Kubernetes project deprecates the FlexVolume volume plugin in Kubernetes version 1.23. See Provisioning PVCs on the Block Volume Service. • By mounting file systems in the Oracle Cloud Infrastructure File Storage service. The File Storage service file systems are mounted inside containers running on clusters created by Kubernetes Engine using a CSI (Container Storage Interface) volume plugin deployed on the clusters. See Provisioning PVCs on the File Storage Service. <p data-bbox="714 1300 1612 1362">https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengcreatingpersistentvolumeclaim.htm</p>

Claim 1	Accused Instrumentalities
	<p data-bbox="730 272 898 310">Overview</p> <p data-bbox="730 362 1892 586">At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.</p> <div data-bbox="743 621 1892 979"> <p>The diagram illustrates the components of an image manifest. On the left, a code block for a Java class is shown: <code>public class HelloWorld { public static void main(String[] args) { System.out.println("Hello, World"); } }</code>. An arrow points from this code to a cylinder labeled 'layer' containing the paths <code>/bin/java</code>, <code>/opt/app.jar</code>, and <code>/lib/libc</code>. This layer is then combined with an 'image index' (represented by a document icon) which contains a JSON structure: <code>{ "manifests": { "platform": { "os": "linux", ... } } }</code>. Finally, this is combined with a 'config' file (also a document icon) containing: <code>{ ... "config": { "Cmd": ["java", "-jar", "app.jar"], ... } }</code>. Each of the three components (layer, image index, and config) has a small square icon with the letters 'ci' in the top-left corner.</p> <p data-bbox="714 1008 1503 1073">https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md</p> </div>

Claim 1	Accused Instrumentalities
	<p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p>
<p>[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p>	<p>In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.</p> <p>For example, in Docker or Kubernetes containers used within the Accused Instrumentalities, each container operates independently, and a base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When an image is used to create a container in the Accused Instrumentality, an instance of the SLCSE is provided to a</p>

Claim 1	Accused Instrumentalities
	<p>software application. Therefore, different instances of the SLCSE are provided to different applications for performing a same function, simultaneously.</p> <p><i>See, e.g.:</i></p> <p>Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container.</p> <p>https://www.techtarget.com/searchitoperations/definition/Docker-image</p> <p>A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.</p> <p>https://docs.docker.com/get-started/overview/</p> <p>Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.</p>  <p>https://docs.docker.com/storage/storagedriver/</p>

JS 44 (Rev. 10/20)

CIVIL COVER SHEET

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. (SEE INSTRUCTIONS ON NEXT PAGE OF THIS FORM.)

I. (a) PLAINTIFFS

VIRTAMOVE, CORP.

(b) County of Residence of First Listed Plaintiff _____
(EXCEPT IN U.S. PLAINTIFF CASES)

(c) Attorneys (Firm Name, Address, and Telephone Number)

Russ August & Kabat, 12424 Wilshire Boulevard, 12th
Floor, Los Angeles, California 90025, Tel.: (310) 826-7474

DEFENDANTS

ORACLE CORPORATION

County of Residence of First Listed Defendant _____
(IN U.S. PLAINTIFF CASES ONLY)

NOTE: IN LAND CONDEMNATION CASES, USE THE LOCATION OF
THE TRACT OF LAND INVOLVED.

Attorneys (If Known)

II. BASIS OF JURISDICTION (Place an "X" in One Box Only)

- ☐ 1 U.S. Government Plaintiff ☒ 3 Federal Question
(U.S. Government Not a Party)
- ☐ 2 U.S. Government Defendant ☐ 4 Diversity
(Indicate Citizenship of Parties in Item III)

III. CITIZENSHIP OF PRINCIPAL PARTIES (Place an "X" in One Box for Plaintiff and One Box for Defendant)

- | | PTF | DEF | | PTF | DEF |
|---|----------------------------|----------------------------|---|----------------------------|----------------------------|
| Citizen of This State | <input type="checkbox"/> 1 | <input type="checkbox"/> 1 | Incorporated or Principal Place of Business In This State | <input type="checkbox"/> 4 | <input type="checkbox"/> 4 |
| Citizen of Another State | <input type="checkbox"/> 2 | <input type="checkbox"/> 2 | Incorporated and Principal Place of Business In Another State | <input type="checkbox"/> 5 | <input type="checkbox"/> 5 |
| Citizen or Subject of a Foreign Country | <input type="checkbox"/> 3 | <input type="checkbox"/> 3 | Foreign Nation | <input type="checkbox"/> 6 | <input type="checkbox"/> 6 |

IV. NATURE OF SUIT (Place an "X" in One Box Only)

Click here for: [Nature of Suit Code Descriptions.](#)

CONTRACT	TORTS	FORFEITURE/PENALTY	BANKRUPTCY	OTHER STATUTES
<input type="checkbox"/> 110 Insurance <input type="checkbox"/> 120 Marine <input type="checkbox"/> 130 Miller Act <input type="checkbox"/> 140 Negotiable Instrument <input type="checkbox"/> 150 Recovery of Overpayment & Enforcement of Judgment <input type="checkbox"/> 151 Medicare Act <input type="checkbox"/> 152 Recovery of Defaulted Student Loans (Excludes Veterans) <input type="checkbox"/> 153 Recovery of Overpayment of Veteran's Benefits <input type="checkbox"/> 160 Stockholders' Suits <input type="checkbox"/> 190 Other Contract <input type="checkbox"/> 195 Contract Product Liability <input type="checkbox"/> 196 Franchise	PERSONAL INJURY <input type="checkbox"/> 310 Airplane <input type="checkbox"/> 315 Airplane Product Liability <input type="checkbox"/> 320 Assault, Libel & Slander <input type="checkbox"/> 330 Federal Employers' Liability <input type="checkbox"/> 340 Marine <input type="checkbox"/> 345 Marine Product Liability <input type="checkbox"/> 350 Motor Vehicle <input type="checkbox"/> 355 Motor Vehicle Product Liability <input type="checkbox"/> 360 Other Personal Injury <input type="checkbox"/> 362 Personal Injury - Medical Malpractice PERSONAL INJURY <input type="checkbox"/> 365 Personal Injury - Product Liability <input type="checkbox"/> 367 Health Care/Pharmaceutical Personal Injury Product Liability <input type="checkbox"/> 368 Asbestos Personal Injury Product Liability PERSONAL PROPERTY <input type="checkbox"/> 370 Other Fraud <input type="checkbox"/> 371 Truth in Lending <input type="checkbox"/> 380 Other Personal Property Damage <input type="checkbox"/> 385 Property Damage Product Liability	<input type="checkbox"/> 625 Drug Related Seizure of Property 21 USC 881 <input type="checkbox"/> 690 Other LABOR <input type="checkbox"/> 710 Fair Labor Standards Act <input type="checkbox"/> 720 Labor/Management Relations <input type="checkbox"/> 740 Railway Labor Act <input type="checkbox"/> 751 Family and Medical Leave Act <input type="checkbox"/> 790 Other Labor Litigation <input type="checkbox"/> 791 Employee Retirement Income Security Act IMMIGRATION <input type="checkbox"/> 462 Naturalization Application <input type="checkbox"/> 465 Other Immigration Actions	<input type="checkbox"/> 422 Appeal 28 USC 158 <input type="checkbox"/> 423 Withdrawal 28 USC 157 PROPERTY RIGHTS <input type="checkbox"/> 820 Copyrights <input checked="" type="checkbox"/> 830 Patent <input type="checkbox"/> 835 Patent - Abbreviated New Drug Application <input type="checkbox"/> 840 Trademark <input type="checkbox"/> 880 Defend Trade Secrets Act of 2016 SOCIAL SECURITY <input type="checkbox"/> 861 HIA (1395ff) <input type="checkbox"/> 862 Black Lung (923) <input type="checkbox"/> 863 DIWC/DIWW (405(g)) <input type="checkbox"/> 864 SSID Title XVI <input type="checkbox"/> 865 RSI (405(g)) FEDERAL TAX SUITS <input type="checkbox"/> 870 Taxes (U.S. Plaintiff or Defendant) <input type="checkbox"/> 871 IRS—Third Party 26 USC 7609	<input type="checkbox"/> 375 False Claims Act <input type="checkbox"/> 376 Qui Tam (31 USC 3729(a)) <input type="checkbox"/> 400 State Reapportionment <input type="checkbox"/> 410 Antitrust <input type="checkbox"/> 430 Banks and Banking <input type="checkbox"/> 450 Commerce <input type="checkbox"/> 460 Deportation <input type="checkbox"/> 470 Racketeer Influenced and Corrupt Organizations <input type="checkbox"/> 480 Consumer Credit (15 USC 1681 or 1692) <input type="checkbox"/> 485 Telephone Consumer Protection Act <input type="checkbox"/> 490 Cable/Sat TV <input type="checkbox"/> 850 Securities/Commodities/Exchange <input type="checkbox"/> 890 Other Statutory Actions <input type="checkbox"/> 891 Agricultural Acts <input type="checkbox"/> 893 Environmental Matters <input type="checkbox"/> 895 Freedom of Information Act <input type="checkbox"/> 896 Arbitration <input type="checkbox"/> 899 Administrative Procedure Act/Review or Appeal of Agency Decision <input type="checkbox"/> 950 Constitutionality of State Statutes
REAL PROPERTY <input type="checkbox"/> 210 Land Condemnation <input type="checkbox"/> 220 Foreclosure <input type="checkbox"/> 230 Rent Lease & Ejectment <input type="checkbox"/> 240 Torts to Land <input type="checkbox"/> 245 Tort Product Liability <input type="checkbox"/> 290 All Other Real Property	CIVIL RIGHTS <input type="checkbox"/> 440 Other Civil Rights <input type="checkbox"/> 441 Voting <input type="checkbox"/> 442 Employment <input type="checkbox"/> 443 Housing/Accommodations <input type="checkbox"/> 445 Amer. w/Disabilities - Employment <input type="checkbox"/> 446 Amer. w/Disabilities - Other <input type="checkbox"/> 448 Education PRISONER PETITIONS Habeas Corpus: <input type="checkbox"/> 463 Alien Detainee <input type="checkbox"/> 510 Motions to Vacate Sentence <input type="checkbox"/> 530 General <input type="checkbox"/> 535 Death Penalty Other: <input type="checkbox"/> 540 Mandamus & Other <input type="checkbox"/> 550 Civil Rights <input type="checkbox"/> 555 Prison Condition <input type="checkbox"/> 560 Civil Detainee - Conditions of Confinement			

V. ORIGIN (Place an "X" in One Box Only)

- ☒ 1 Original Proceeding ☐ 2 Removed from State Court ☐ 3 Remanded from Appellate Court ☐ 4 Reinstated or Reopened ☐ 5 Transferred from Another District (specify) ☐ 6 Multidistrict Litigation - Transfer ☐ 8 Multidistrict Litigation - Direct File

VI. CAUSE OF ACTION

Cite the U.S. Civil Statute under which you are filing (Do not cite jurisdictional statutes unless diversity):
35 U.S.C. § 1 et seq.

Brief description of cause:
Patent Infringement

VII. REQUESTED IN COMPLAINT:

☐ CHECK IF THIS IS A CLASS ACTION UNDER RULE 23, F.R.Cv.P.

DEMAND \$

CHECK YES only if demanded in complaint:

JURY DEMAND: ☒ Yes ☐ No

VIII. RELATED CASE(S) IF ANY

(See instructions):

JUDGE Counts; Albright

DOCKET NUMBER 7:24-cv-00030; 7:24-cv-00033

DATE

Dec 20, 2024

SIGNATURE OF ATTORNEY OF RECORD

/s/ Reza Mirzaie

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG. JUDGE _____

INSTRUCTIONS FOR ATTORNEYS COMPLETING CIVIL COVER SHEET FORM JS 44

Authority For Civil Cover Sheet

The JS 44 civil cover sheet and the information contained herein neither replaces nor supplements the filings and service of pleading or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. Consequently, a civil cover sheet is submitted to the Clerk of Court for each civil complaint filed. The attorney filing a case should complete the form as follows:

- I.(a) Plaintiffs-Defendants.** Enter names (last, first, middle initial) of plaintiff and defendant. If the plaintiff or defendant is a government agency, use only the full name or standard abbreviations. If the plaintiff or defendant is an official within a government agency, identify first the agency and then the official, giving both name and title.
 - (b) County of Residence.** For each civil case filed, except U.S. plaintiff cases, enter the name of the county where the first listed plaintiff resides at the time of filing. In U.S. plaintiff cases, enter the name of the county in which the first listed defendant resides at the time of filing. (NOTE: In land condemnation cases, the county of residence of the "defendant" is the location of the tract of land involved.)
 - (c) Attorneys.** Enter the firm name, address, telephone number, and attorney of record. If there are several attorneys, list them on an attachment, noting in this section "(see attachment)".
- II. Jurisdiction.** The basis of jurisdiction is set forth under Rule 8(a), F.R.Cv.P., which requires that jurisdictions be shown in pleadings. Place an "X" in one of the boxes. If there is more than one basis of jurisdiction, precedence is given in the order shown below.
- United States plaintiff. (1) Jurisdiction based on 28 U.S.C. 1345 and 1348. Suits by agencies and officers of the United States are included here. United States defendant. (2) When the plaintiff is suing the United States, its officers or agencies, place an "X" in this box.
- Federal question. (3) This refers to suits under 28 U.S.C. 1331, where jurisdiction arises under the Constitution of the United States, an amendment to the Constitution, an act of Congress or a treaty of the United States. In cases where the U.S. is a party, the U.S. plaintiff or defendant code takes precedence, and box 1 or 2 should be marked.
- Diversity of citizenship. (4) This refers to suits under 28 U.S.C. 1332, where parties are citizens of different states. When Box 4 is checked, the citizenship of the different parties must be checked. (See Section III below; **NOTE: federal question actions take precedence over diversity cases.**)
- III. Residence (citizenship) of Principal Parties.** This section of the JS 44 is to be completed if diversity of citizenship was indicated above. Mark this section for each principal party.
- IV. Nature of Suit.** Place an "X" in the appropriate box. If there are multiple nature of suit codes associated with the case, pick the nature of suit code that is most applicable. Click here for: [Nature of Suit Code Descriptions](#).
- V. Origin.** Place an "X" in one of the seven boxes.
- Original Proceedings. (1) Cases which originate in the United States district courts.
- Removed from State Court. (2) Proceedings initiated in state courts may be removed to the district courts under Title 28 U.S.C., Section 1441.
- Remanded from Appellate Court. (3) Check this box for cases remanded to the district court for further action. Use the date of remand as the filing date.
- Reinstated or Reopened. (4) Check this box for cases reinstated or reopened in the district court. Use the reopening date as the filing date.
- Transferred from Another District. (5) For cases transferred under Title 28 U.S.C. Section 1404(a). Do not use this for within district transfers or multidistrict litigation transfers.
- Multidistrict Litigation – Transfer. (6) Check this box when a multidistrict case is transferred into the district under authority of Title 28 U.S.C. Section 1407.
- Multidistrict Litigation – Direct File. (8) Check this box when a multidistrict case is filed in the same district as the Master MDL docket.
- PLEASE NOTE THAT THERE IS NOT AN ORIGIN CODE 7.** Origin Code 7 was used for historical records and is no longer relevant due to changes in statute.
- VI. Cause of Action.** Report the civil statute directly related to the cause of action and give a brief description of the cause. **Do not cite jurisdictional statutes unless diversity.** Example: U.S. Civil Statute: 47 USC 553 Brief Description: Unauthorized reception of cable service.
- VII. Requested in Complaint.** Class Action. Place an "X" in this box if you are filing a class action under Rule 23, F.R.Cv.P.
- Demand. In this space enter the actual dollar amount being demanded or indicate other demand, such as a preliminary injunction.
- Jury Demand. Check the appropriate box to indicate whether or not a jury is being demanded.
- VIII. Related Cases.** This section of the JS 44 is used to reference related pending cases, if any. If there are related pending cases, insert the docket numbers and the corresponding judge names for such cases.

Date and Attorney Signature. Date and sign the civil cover sheet.

EXHIBIT O

Brandon H. Brown (SBN 266347)
Kyle Calhoun (SBN 311181)
KIRKLAND & ELLIS LLP
555 California Street
San Francisco, CA 94104
Telephone: (415) 439-1400
Facsimile: (415) 439-1500
Email: brandon.brown@kirkland.com
Email: kyle.calhoun@kirkland.com

Todd M. Friedman (admitted *pro hac vice*)
KIRKLAND & ELLIS LLP
601 Lexington Avenue
New York, NY 10022
Telephone: (212) 446-4800
Facsimile: (212) 446-4900
Email: todd.friedman@kirkland.com

Attorneys for Plaintiff RED HAT, INC.

**UNITED STATES DISTRICT COURT
NORTHERN DISTRICT OF CALIFORNIA
SAN JOSE DIVISION**

RED HAT, INC.,

Plaintiff,

v.

VIRTAMOVE, CORP.,

Defendant.

CASE NO. 5:24-CV-04740-PCP

**DECLARATION OF MICHAEL
BARRETT IN SUPPORT OF
PLAINTIFF RED HAT, INC.'S
MOTION FOR LEAVE TO
SUPPLEMENT ITS COMPLAINT**

Location: Courtroom 8, 4th Floor
Judge: Hon. P. Casey Pitts

I, Michael Barrett, declare:

1. I am a Vice President and General Manager of Red Hat Hybrid Platforms at Red Hat, Inc. (“Red Hat”), where I have been employed since 2013. In this role, I am responsible for product management, product marketing, technical marketing and enablement, and strategic partnerships across Red Hat’s Hybrid Platforms business unit. These responsibilities include managing Red Hat’s partnerships and interactions with cloud providers such as Microsoft and Oracle. I am aware of the containerization and container orchestration products sold or supported by Microsoft and Oracle, including their products that include or interact with Red Hat’s OpenShift products. I either have personal knowledge of the facts contained in this declaration and could competently testify to the matters set forth herein if called upon to do so.

2. I understand that Red Hat has filed a declaratory judgment suit in California against VirtaMove, Corp. (“VirtaMove”). I also understand that VirtaMove recently filed suit against Microsoft Corporation (“Microsoft”) and Oracle Corporation (“Oracle”) in Texas.

3. Azure Red Hat OpenShift (“ARO”) is one of Microsoft Azure’s cloud service offerings that provides for the orchestration of containerized applications using Red Hat’s OpenShift. ARO is jointly engineered, operated, managed, and supported by Red Hat and Microsoft as part of a partnership first established in 2018. Through ARO, Microsoft offers Red Hat OpenShift product directly to the customers, who purchase ARO through their accounts with Microsoft. Revenues from ARO sales are distributed between Red Hat and Microsoft. ARO is derived from the same open source Kubernetes containerization technology as Microsoft’s other container solutions available via the Azure computing system, such as Azure Kubernetes Service (“AKS”).

4. The self-managed Red Hat OpenShift Container Platform is a containerized application product that can be deployed on the Oracle Cloud Infrastructure (“OCI”). OCI is Oracle’s cloud services infrastructure that allows users to build and run applications and services in a high-availability, hosted

environment. Red Hat's OpenShift, including its Kubernetes-based cluster workloads and container orchestration features, can be and has been deployed and run on Oracle's OCI platform. Other Oracle services, such as Oracle Kubernetes Engine offer similar containerized application products, and like OpenShift, are also based on Kubernetes.

I declare under penalty of perjury that the foregoing is true and correct to the best of my knowledge.

Executed this 10 day of January, 2025, in Cincinnati, Ohio.


Michael Barrett

EXHIBIT P

Azure Red Hat OpenShift

Article • 07/10/2024

The Microsoft *Azure Red Hat OpenShift* service enables you to deploy fully managed [OpenShift](#) clusters. Azure Red Hat OpenShift extends [Kubernetes](#) . Running containers in production with Kubernetes requires additional tools and resources. This often includes needing to juggle image registries, storage management, networking solutions, and logging and monitoring tools - all of which must be versioned and tested together. Building container-based applications requires even more integration work with middleware, frameworks, databases, and CI/CD tools. Azure Red Hat OpenShift combines all this into a single platform, bringing ease of operations to IT teams while giving application teams what they need to execute.

Azure Red Hat OpenShift is jointly engineered, operated, and supported by Red Hat and Microsoft to provide an integrated support experience. There are no virtual machines to operate, and no patching is required. Control plane, infrastructure, and application nodes are patched, updated, and monitored on your behalf by Red Hat and Microsoft. Your Azure Red Hat OpenShift clusters are deployed into your Azure subscription and are included on your Azure bill.

You can choose your own registry, networking, storage, and CI/CD solutions, or use the built-in solutions for automated source code management, container and application builds, deployments, scaling, health management, and more. Azure Red Hat OpenShift provides an integrated sign-on experience through Microsoft Entra ID.

Access, security, and monitoring

For improved security and management, Azure Red Hat OpenShift lets you integrate with Microsoft Entra ID and use Kubernetes role-based access control (Kubernetes RBAC). You can also monitor the health of your cluster and resources.

Cluster and node

Azure Red Hat OpenShift nodes run on Azure virtual machines. You can connect storage to nodes and pods and upgrade cluster components.

Service Level Agreement

Azure Red Hat OpenShift offers a Service Level Agreement to guarantee that the service will be available 99.95% of the time. For more details on the SLA, see [Azure Red Hat OpenShift SLA](#) .

1/9/25, 9:20 AM

Introduction to Azure Red Hat OpenShift - Azure Red Hat OpenShift | Microsoft Learn

Next steps

Learn the prerequisites for Azure Red Hat OpenShift:

Create an OpenShift Cluster

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

EXHIBIT Q

Microsoft | Azure Explore Products Solutions More Show search input Learn Support Contact Sales Get started with Azure Sign in

Azure Red Hat OpenShift is now available in Azure Government

Azure Red Hat OpenShift

Deploy and scale containers on managed Red Hat OpenShift

Get started with Azure

Overview Capabilities Security Pricing Customer stories Resources FAQ Next steps Get started with Azure

OVERVIEW

Improve DevOps with a fully managed, turnkey container platform

Benefits Less complex delivery

Build, deploy, and scale apps on OpenShift with confidence

Azure Red Hat OpenShift provides highly available, fully managed OpenShift clusters on demand, monitored and operated jointly by Microsoft and Red Hat. Kubernetes is at the core of Red Hat OpenShift. OpenShift brings value-added features to complement Kubernetes, making it a turnkey container platform as a service (PaaS) that significantly improves developer and operator experiences.

[Explore documentation](#)

platform as a service (PaaS) that significantly improves developer and operator experiences.

[Explore documentation](#)

Focus on what matters ▾

Provision self-service, on-demand application stacks ▾

CAPABILITIES

Key features



Fully managed clusters

Highly available, fully managed public and private clusters, automated operations, and over-the-air platform upgrades.



Unified operations

Built on top of upstream Kubernetes and packaged with key capabilities for unified operations as a complete platform.



Joint services and support

Joint engineering, operation, and support from Microsoft and Red Hat, with multiple compliance certifications.



Deploy clusters globally

Clusters deployed to Azure and billed to your subscription in more than 20 regions worldwide.

SECURITY

Built-in security and compliance

Microsoft has committed to [investing \\$20 billion in cybersecurity](#) over five years.

We employ more than [8,500 security and threat intelligence experts](#) across 77 countries.

Azure has one of the largest [compliance certification portfolios](#) in the industry.

[Learn more about security on Azure](#)



AI-powered assistant



Microsoft has committed to [investing \\$20 billion in cybersecurity](#) over five years.

We employ more than [8,500 security and threat intelligence experts](#) across 77 countries.

Azure has one of the largest [compliance certification portfolios](#) in the industry.

[Learn more about security on Azure](#)

PRICING

Flexible and on-demand pricing

Azure Red Hat OpenShift charges for the virtual machines (VMs) provisioned in the clusters and OpenShift licenses based on the VM instance selected. Deploy a cluster using on-demand pricing or purchase OpenShift worker node reserved instances, whichever best meets the needs of your workload and business.

[See pricing](#)

[Purchase reserved instances](#)



CUSTOMER STORIES

Customers are doing great things on Azure Red Hat OpenShift

[AI-powered assistant](#)



Logistics software company improves sustainability in the transport industry

Alpega's Supply Chain Optimization system, built on Azure Red Hat OpenShift, allows the





Logistics software company improves sustainability in the transport industry

Alpega Group's Smart Booking system, built on containers, allows the company to respond to demand and scale its operations accordingly. This transformation is helping Alpega to optimize freight transportation and create more sustainability within the industry.

[Read the story](#)



RESOURCES

Azure Red Hat OpenShift resources and documentation

Learning resources

Popular developer resources



Intro to Azure Red Hat OpenShift

[Read the article](#)



Regional availability

[See regional availability](#)

AI-powered assistant



St

[Read the article](#)[See regional availability](#)

FAQ

Frequently asked questions

Expand all

Collapse all

01/ How does Red Hat OpenShift compare to Kubernetes?



02/ Can I switch from using Kubernetes to Red Hat OpenShift?



03/ Do I need a separate contract with Red Hat to use the service?



04/ Who should I contact for support?



05/ Where is Azure Red Hat OpenShift available?



06/ What is the SLA?



AI-powered assistant



Next Steps

Choose the Azure account that's right for you



Next Steps

Choose the Azure account that's right for you

Pay as you go or try Azure free for up to 30 days.

[Get started with Azure](#)


Azure Solutions

Azure cloud solutions

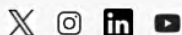
Solve your business problems with proven combinations of Azure cloud services, as well as sample architectures and documentation.

[Explore Azure solutions](#)


Business Solution Hub

Find the right Microsoft Cloud solution

Browse the Microsoft Business Solutions Hub to find the products and solutions that can help your organization reach its goals.

[Explore Microsoft solutions](#)
[Get the Azure mobile app](#)


Explore Azure

What is Azure?
Get started with Azure
Global infrastructure
Datacenter regions
Trust your cloud
Azure Essentials
Customer stories

Products and pricing

Products
Azure pricing
Free Azure services
Flexible purchase options
FinOps on Azure
Optimize your costs

Solutions and support

Solutions
Resources for accelerating growth
Solution architectures
Support
Azure demo and live Q&A

Partners

Azure Marketplace
Find a partner
Join ISV Success

Resources

Training and certifications
Documentation
Blog
Developer resources
Students
Events and Webinars
Analyst reports, white papers, and e-books
Videos

[AI-powered assistant](#)

[Discover AI-powered assistant](#)

What is cloud computing?
What is multicloud?
What is machine learning?
What is deep learning?
What is AlaaS?
What are LLMs?
What are SLMs?
What is RAG?

EXHIBIT R

What is Azure Kubernetes Service (AKS)?

Article • 12/04/2024

Azure Kubernetes Service (AKS) is a managed Kubernetes service that you can use to deploy and manage containerized applications. You need minimal container orchestration expertise to use AKS. AKS reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure. AKS is an ideal platform for deploying and managing containerized applications that require high availability, scalability, and portability, and for deploying applications to multiple regions, using open-source tools, and integrating with existing DevOps tools.

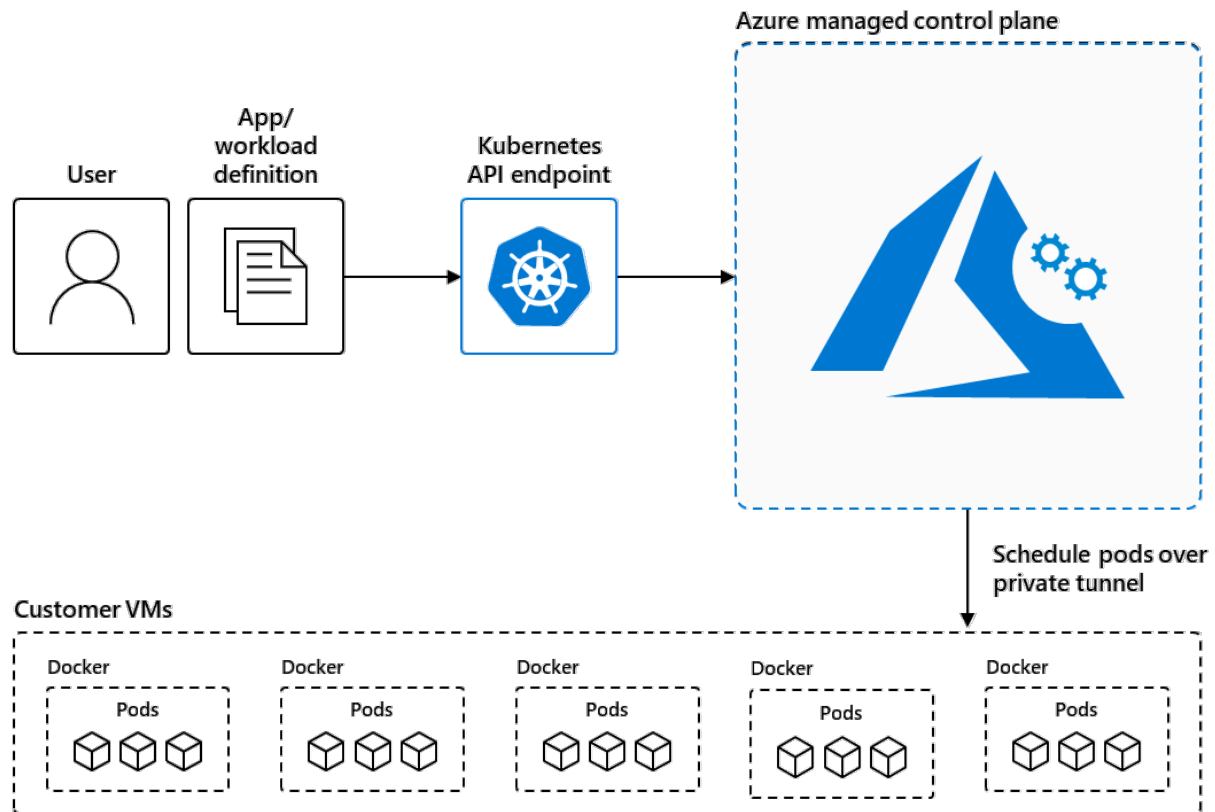
This article is intended for platform administrators or developers who are looking for a scalable, automated, managed Kubernetes solution.

Overview of AKS

AKS reduces the complexity and operational overhead of managing Kubernetes by shifting that responsibility to Azure. When you create an AKS cluster, Azure automatically creates and configures a control plane for you at no cost. The Azure platform manages the AKS control plane, which is responsible for the Kubernetes objects and worker nodes that you deploy to run your applications. Azure takes care of critical operations like health monitoring and maintenance, and you only pay for the AKS nodes that run your applications.

1/9/25, 9:44 AM

What is Azure Kubernetes Service (AKS)? - Azure Kubernetes Service | Microsoft Learn



ⓘ Note

AKS is [CNCF-certified](#) and is compliant with SOC, ISO, PCI DSS, and HIPAA. For more information, see the [Microsoft Azure compliance overview](#).

Container solutions in Azure

Azure offers a range of container solutions designed to accommodate various workloads, architectures, and business needs.

[Expand table](#)

Container solution	Resource type
Azure Kubernetes Service	Managed Kubernetes
Azure Red Hat OpenShift	Managed Kubernetes
Azure Arc-enabled Kubernetes	Unmanaged Kubernetes
Azure Container Instances	Managed Docker container instance

Container solution	Resource type
Azure Container Apps	Managed Kubernetes

For more information comparing the various solutions, see the following resources:

- [Comparing the service models of Azure container solutions](#)
- [Comparing Azure compute service options](#)

When to use AKS

The following list describes some common use cases for AKS:

- **Lift and shift to containers with AKS:** Migrate existing applications to containers and run them in a fully managed Kubernetes environment.
- **Microservices with AKS:** Simplify the deployment and management of microservices-based applications with streamlined horizontal scaling, self-healing, load balancing, and secret management.
- **Secure DevOps for AKS:** Efficiently balance speed and security by implementing secure DevOps with Kubernetes.
- **Bursting from AKS with ACI:** Use virtual nodes to provision pods inside ACI that start in seconds and scale to meet demand.
- **Machine learning model training with AKS:** Train models using large datasets with familiar tools, such as TensorFlow and Kubeflow.
- **Data streaming with AKS:** Ingest and process real-time data streams with millions of data points collected via sensors, and perform fast analyses and computations to develop insights into complex scenarios.
- **Using Windows containers on AKS:** Run Windows Server containers on AKS to modernize your Windows applications and infrastructure.

Features of AKS

The following table lists some of the key features of AKS:

 Expand table

Feature	Description
Identity and security management	<ul style="list-style-type: none"> • Enforce regulatory compliance controls using Azure Policy with built-in guardrails and internet security benchmarks. • Integrate with Kubernetes RBAC to limit access to cluster resources.

1/9/25, 9:44 AM

What is Azure Kubernetes Service (AKS)? - Azure Kubernetes Service | Microsoft Learn

Feature	Description
	<ul style="list-style-type: none"> Use Microsoft Entra ID to set up Kubernetes access based on existing identity and group membership.
Logging and monitoring	<ul style="list-style-type: none"> Integrate with Container Insights, a feature in Azure Monitor, to monitor the health and performance of your clusters and containerized applications. Set up Advanced Container Networking Services to collect and visualize network traffic data from your clusters.
Streamlined deployments	<ul style="list-style-type: none"> Use prebuilt cluster configurations for Kubernetes with smart defaults. Autoscale your applications using the Kubernetes Event Driven Autoscaler (KEDA). Use Draft for AKS to ready source code and prepare your applications for production.
Clusters and nodes	<ul style="list-style-type: none"> Connect storage to nodes and pods, upgrade cluster components, and use GPUs. Create clusters that run multiple node pools to support mixed operating systems and Windows Server containers. Configure automatic scaling using the cluster autoscaler and horizontal pod autoscaler. Deploy clusters with confidential computing nodes to allow containers to run in a hardware-based trusted execution environment.
Storage volume support	<ul style="list-style-type: none"> Mount static or dynamic storage volumes for persistent data. Use Azure Container Storage for fully managed, cloud-based volume management and orchestration of block storage. Azure Container Storage integrates with Kubernetes, allowing dynamic and automatic provisioning of persistent volumes. Use Azure Disks CSI driver for single pod access and Azure Files CSI driver for multiple, concurrent pod access. Use Azure NetApp Files for high-performance, high-throughput, and low-latency file shares.
Networking	<ul style="list-style-type: none"> Leverage our networking options for your needs. Bring your own Container Network Interface (CNI) to use a third-party CNI plugin. Easily access applications deployed to your clusters using the application routing add-on with nginx.
Development tooling integration	<ul style="list-style-type: none"> Develop on AKS with Helm. Install the Kubernetes extension for Visual Studio Code to manage your workloads. Leverage the features of Istio with the Istio-based service mesh add-on.

Get started with AKS

Get started with AKS using the following resources:

- Learn the [core Kubernetes concepts for AKS](#).
- Evaluate application deployment on AKS with our [AKS tutorial series](#).

1/9/25, 9:44 AM

What is Azure Kubernetes Service (AKS)? - Azure Kubernetes Service | Microsoft Learn

- Review the [Azure Well-Architected Framework for AKS](#) to learn how to design and operate reliable, secure, efficient, and cost-effective applications on AKS.
- [Plan your design and operations](#) for AKS using our reference architectures.
- Explore [configuration options and recommended best practices for cost optimization](#) on AKS.

EXHIBIT S

1/9/25, 9:46 AM

Red Hat OpenShift Generally Available on Oracle Cloud Infrastructure

ORACLE

Products Industries Resources Customers Partners Developers Company



View Accounts

Contact Sales

Press Release

Red Hat OpenShift Generally Available on Oracle Cloud Infrastructure

Expanded collaboration enables validated and supported configurations of Red Hat OpenShift to run on OCI

Combines the leading hybrid cloud application platform with the leading distributed cloud

Red Hat Summit 2024, Denver and Austin, Texas—May 6, 2024



Red Hat, Inc., the world's leading provider of open source solutions, and Oracle today announced the general availability of [Red Hat OpenShift](#) on [Oracle Cloud Infrastructure](#) (OCI) Compute Virtual Machines (VMs). Red Hat OpenShift is the industry's leading hybrid cloud application platform powered by

<https://www.oracle.com/news/announcement/red-hat-openshift-generally-available-on-oracle-cloud-infrastructure-2024-05-06/>

1/5

1/9/25, 9:46 AM

Red Hat OpenShift Generally Available on Oracle Cloud Infrastructure

Kubernetes for architecting, building, and deploying cloud-native applications. Offering a distributed cloud of 69 regions and counting, OCI can run Red Hat OpenShift in the location and operations model that best meets customer needs for regulatory compliance, performance, and cost-effectiveness. The new offering builds on the collaboration between Red Hat and Oracle that began with the certification of Red Hat Enterprise Linux for OCI bare metal and Oracle Cloud VMware Solution workloads.

The combination of [Red Hat Enterprise Linux](#) certification and Red Hat OpenShift gives customers the confidence to install, migrate, and run Red Hat OpenShift workloads on OCI, supported by existing and transparent support agreements between Red Hat and Oracle. Red Hat OpenShift on OCI is supported for customer-managed installations using validated configurations of Red Hat OpenShift Platform Plus, Red Hat OpenShift Container Platform, and Red Hat OpenShift Kubernetes Engine.

Customers can now extend their Red Hat OpenShift ecosystem to include installations on OCI, managed from their Red Hat portal. Customers can choose from multiple installation methods, including Red Hat OpenShift Assisted Installer, command line, and agent-based, which enables installation in air-gapped environments. Oracle is providing Container Storage Interface (CSI) software that enables OCI storage integration with Red Hat OpenShift, and Cloud Control Manager (CCM) software that enables API interoperation between OCI and Red Hat OpenShift platform.

OCI's [distributed cloud](#) includes Oracle Government Cloud regions in the U.S., U.K., and Australia; OCI Dedicated Regions at customer-controlled sites; partner-enabled Oracle Alloy regions; Compute Cloud@Customer in customers' data centers; and Oracle EU Sovereign Cloud. Red Hat OpenShift versions 4.14 and 4.15 are validated for installation on OCI Compute flexible virtual machine shapes, available in each of these offerings. This deployment flexibility is especially critical for organizations in industries with complex regulatory environments such as telecommunications, finance, and healthcare, as well as organizations operating across multiple jurisdictions. OCI Compute flexible virtual machine shapes and OCI Block Storage auto-tuning volumes optimize performance at one-half the compute and one-third the storage prices of other hyperscalers, respectively. Bare metal validation is planned in the future.

Learn more about Red Hat OpenShift on OCI at the [Red Hat Summit](#) on May 6-9 in Denver, Colorado.

About Red Hat OpenShift

Red Hat OpenShift, the industry's leading hybrid cloud application platform powered by Kubernetes, brings together tested and trusted services to reduce the friction of developing, modernizing, deploying, running and managing applications—delivering a consistent experience across public cloud, on-premise, hybrid cloud, or edge architecture. Red Hat OpenShift is available as a turnkey application platform from major cloud providers to empower customers with the flexibility to modernize and scale in the cloud of their choice.

Red Hat Summit

Join the Red Hat Summit keynotes to hear the latest from Red Hat executives, customers and partners:

[The cloud is hybrid. So is AI.](#) — Tuesday, May 7, 8-10 a.m. MDT ([YouTube](#), [LinkedIn](#))

[Optimizing IT for the AI era.](#) — Wednesday, May 8, 8:30-9:30 a.m. MDT ([YouTube](#), [LinkedIn](#))

Supporting Quotes

1/9/25, 9:46 AM

Red Hat OpenShift Generally Available on Oracle Cloud Infrastructure

Ashesh Badani, senior vice president and chief product officer, Red Hat

“Flexibility and choice are paramount needs for organizations as they build, deploy, and manage enterprise applications across the hybrid cloud. By extending Red Hat OpenShift to OCI, we’re providing our customers with the ability to run their applications in the environment that best meets their unique needs, including the higher performance, enhanced security, and greater cost-effectiveness of OCI.”

Karan Batta, senior vice president, Oracle Cloud Infrastructure

“As enterprises continue to leverage multicloud environments to help meet their business needs and goals, they are prioritizing flexibility and simplicity as they migrate their workloads to the cloud. With Red Hat OpenShift on OCI, customers achieve the double benefit of easily extending their Red Hat OpenShift environments to OCI, while gaining the flexibility to run their workloads from any location via OCI’s distributed cloud.”

Additional Resources

Try [Oracle Cloud today](#)

Learn more about [Oracle Cloud Infrastructure](#)

Learn more about [Red Hat OpenShift](#) on OCI

Learn more about [Red Hat Enterprise Linux](#) running on OCI bare metal instances

See all of Red Hat’s news announcements this week in the [Red Hat Summit newsroom](#)

Follow [@RedHatSummit](#) or [#RHSummit](#) on Twitter/X for event-specific updates

Connect with Red Hat

Learn more about [Red Hat](#)

Get more news in the [Red Hat newsroom](#)

Read the [Red Hat blog](#)

Follow [Red Hat on X/Twitter](#)

Join [Red Hat on Instagram](#)

Watch [Red Hat videos on YouTube](#)

Follow [Red Hat on LinkedIn](#)

Contact Info

Drew Smith

Oracle

drew.j.smith@oracle.com

415.336.1103

Jessie Beach

Red Hat

jbeach@redhat.com

1/9/25, 9:46 AM

Red Hat OpenShift Generally Available on Oracle Cloud Infrastructure

+1-919-602-2836

About Red Hat, Inc.

Red Hat is the world's leading provider of enterprise open source software solutions, using a community-powered approach to deliver reliable and high-performing Linux, hybrid cloud, container, and Kubernetes technologies. Red Hat helps customers integrate new and existing IT applications, develop cloud-native applications, standardize on our industry-leading operating system, and automate, secure, and manage complex environments.. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500. As a strategic partner to cloud providers, system integrators, application vendors, customers, and open source communities, Red Hat can help organizations prepare for the digital future.

About Oracle

Oracle offers integrated suites of applications plus secure, autonomous infrastructure in the Oracle Cloud. For more information about Oracle (NYSE: ORCL), please visit us at www.oracle.com.

Red Hat Forward-Looking Statements

Except for the historical information and discussions contained herein, statements contained in this press release may constitute forward-looking statements within the meaning of the Private Securities Litigation Reform Act of 1995. Forward-looking statements are based on the company's current assumptions regarding future business and financial performance. These statements involve a number of risks, uncertainties and other factors that could cause actual results to differ materially. Any forward-looking statement in this press release speaks only as of the date on which it is made. Except as required by law, the company assumes no obligation to update or revise any forward-looking statements.

Oracle Forward-Looking Statements Disclaimer

Statements in this article relating to Oracle's future plans, expectations, beliefs, and intentions are "forward-looking statements" and are subject to material risks and uncertainties. Many factors could affect Oracle's current expectations and actual results, and could cause actual results to differ materially. A discussion of such factors and other risks that affect Oracle's business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including Oracle's most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at <http://www.oracle.com/investor>. All information in this article is current as of May 6, 2024 and Oracle undertakes no duty to update any statement in light of new information or future events.

Oracle Future Product Disclaimer

1/9/25, 9:46 AM

Red Hat OpenShift Generally Available on Oracle Cloud Infrastructure

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Trademarks

Oracle, Java, MySQL and NetSuite are registered trademarks of Oracle Corporation. NetSuite was the first cloud company—ushering in the new era of cloud computing.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo and OpenShift are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the U.S. and other countries. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Resources for	Why Oracle	Learn	News and Events	Contact Us
Careers	Analyst Reports	What is cloud computing?	News	US Sales: +1.800.633.0738
Developers	Best cloud-based ERP	What is CRM?	Oracle CloudWorld	How can we help?
Investors	Cloud Economics	What is Docker?	Oracle CloudWorld Tour	Subscribe to emails
Partners	Social Impact	What is Kubernetes?	Oracle Health Summit	Integrity Helpline
Researchers	Culture and Inclusion	What is Python?	Oracle DevLive	Accessibility
Students and Educators	Security Practices	What is SaaS?	Search all events	

EXHIBIT T



Oracle Cloud Infrastructure Documentation

Try Free Tier

[Infrastructure Services](#) >

≡ All Pages

Updated 2024-10-22

Getting Started with OpenShift Container Platform on OCI

Learn about deploying the Red Hat OpenShift Container Platform on Oracle Cloud Infrastructure (OCI).

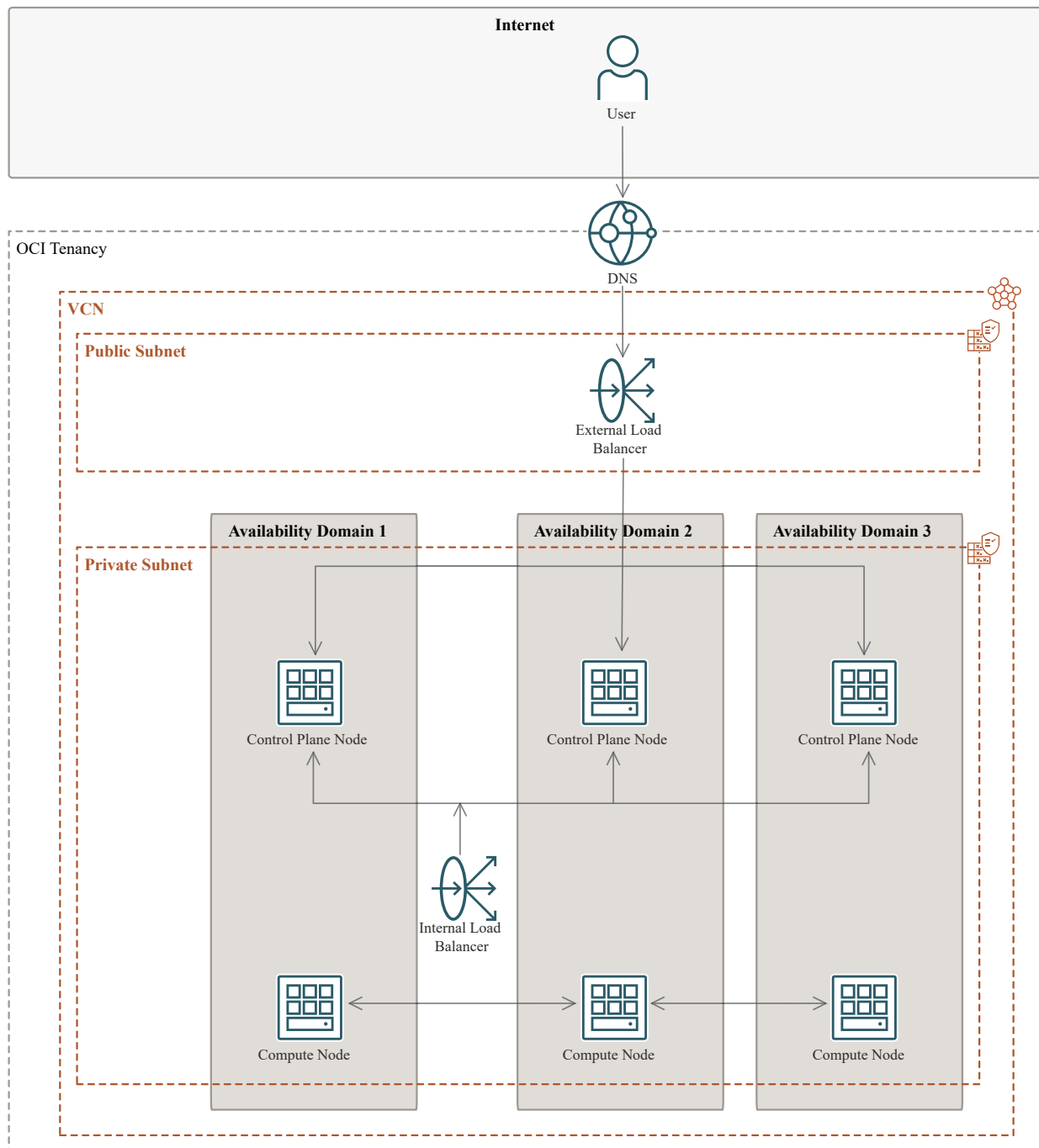
Red Hat OpenShift Container Platform is a cloud-based Kubernetes container platform. Red Hat, in partnership with OCI, supports running cluster workloads on the OCI platform. For an overview of OpenShift Container Platform and Kubernetes, see the Red Hat documentation at [OpenShift Container Platform overview](#) ↗ and [Kubernetes overview](#) ↗.

Cluster infrastructure consists of several compute instances running Red Hat Enterprise Linux CoreOS (RHCOS), along with the networking, load balancing and routing required to move network traffic in and out of the cluster. See [Understanding OpenShift Container Platform](#) ↗ for details about the Red Hat software stack and cluster management.

Cluster Architecture in OCI

OpenShift Container Platform clusters use OCI's DNS Resolution, Virtual Cloud Network (VCN), load balancers, and compute nodes as follows:

1. Network traffic is resolved with OCI DNS.
2. Traffic is routed to the VCN assigned to the cluster compute nodes.
3. Within the VCN's public subnet, an external Load Balancer routes traffic to the control plane (master) nodes of the cluster, which sit within a private subnet.
4. The cluster's control plane compute nodes use an internal Load Balancer to communicate with the compute (worker) nodes of the cluster.



Installation Options

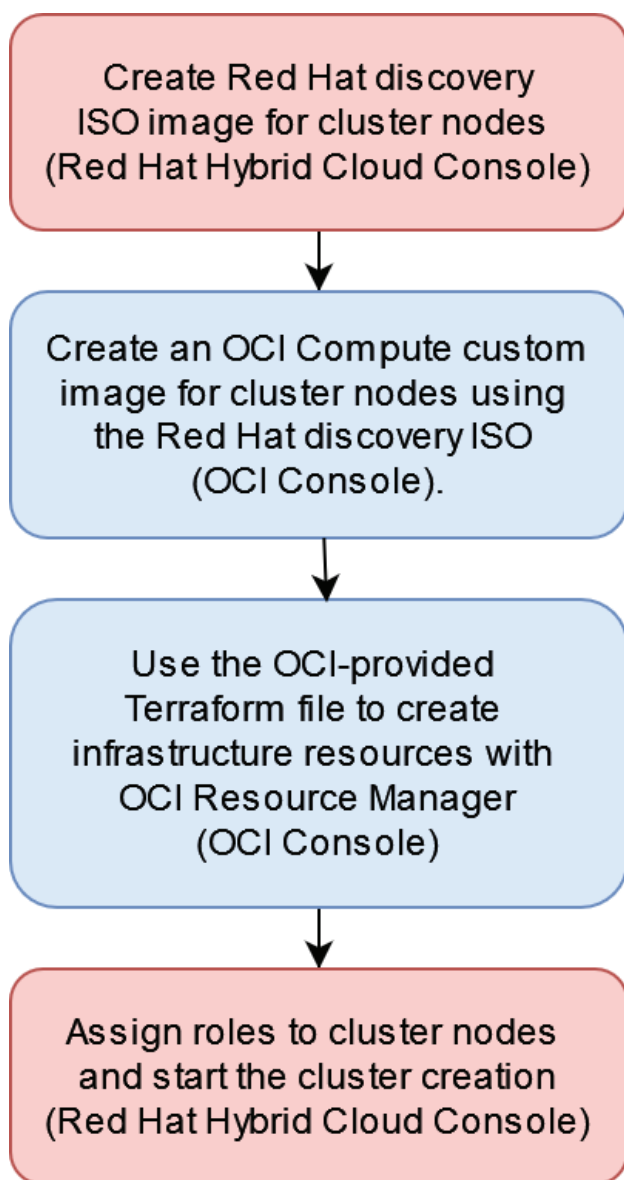
Oracle offers an automated path for provisioning the cluster infrastructure that uses the Red Hat Assisted Installer. We recommend this method for most users.

Oracle also supports the use of Red Hat's Agent-based Installer for users who want to set up the cluster manually or using other automation tools, or for those working in a disconnected environment.

Assisted Installer

Red Hat's Assisted Installer provides a simple web interface in the Red Hat Hybrid Cloud Console to perform cluster installation. The Assisted Installer requires an internet connection.

At a high level, the work flow starts in the Red Hat Hybrid Cloud Console, where you create the discovery ISO image. The work flow then moves to the OCI Console for infrastructure provisioning, which includes creating a custom Red Hat ISO image for compute nodes, and using the Resource Manager service to run a Terraform script and create the infrastructure resources required by the cluster. Lastly, the final cluster configuration and installation is performed in the Red Hat Hybrid Cloud Console:



See [Installing a Cluster with Red Hat's Assisted Installer](#) for instructions.

Agent-based Installer

Red Hat's Agent-based Installer is recommended for advanced users who want maximum flexibility, and requires users to create OCI resources manually in the OCI Console or to use their own automation tools. This method typically takes longer to complete than the Assisted Installer. The work flow is described in detail in [Agent-based Installer for OpenShift Container Platform](#).

Documentation

Deploying an OpenShift Container Platform cluster on OCI combines actions performed in the Red Hat Hybrid Cloud Console and actions performed in the OCI Console. This documentation details the tasks that you perform in the OCI Console, and provides an overview of the complete installation work flow involving both consoles. This documentation is intended to be used together with the Red Hat documentation in the following locations:

- [Installing a cluster on Oracle Cloud Infrastructure \(OCI\) by using the Assisted Installer ↗](#)
- [Installing a cluster on Oracle Cloud Infrastructure \(OCI\) by using the Agent-based Installer ↗](#)

Supported Shapes

Red Hat Enterprise Linux (RHEL) is supported on the virtual machine shapes listed at [Red Hat Ecosystem Catalog - Oracle Cloud Infrastructure ↗](#).

Was this article helpful?

